Dense Graph Pattern Mining and Visualization

Wang Nan

A THESIS SUBMITTED FOR THE DEGREE OF

Doctor of Philosophy

2011

School of Computing

The National University of Singapore

Acknowledgments

I would like to thank my supervisor Associate Professor Anthony K. H. Tung for his guidance on all my work during my PhD candidature, his guidance on how to be a better researcher, and his suggestions on how to be a better person. I would like to thank Professor Kian-Lee Tan and Professor Srinivasan Parthasarathy for their guidance and contribution to the work on CSV: Cohesive Subgraph Mining. I would like to thank Professor Kian-Lee Tan (again) and Mr. Jingbo Zhang for their significant contribution to the work on Triangulation-based Dense Neighborhood Graphs Discovery.

Dedication

To my parents, who offered me unconditional love and support throughout the course of this thesis.

To my husband, Hongjun. Without him, I wouldn't have the courage and strength to finish this thesis.

To my cherish friends, Xiaoli Hu, He Shen, Bingtian and many more. They share my joy and pain.

Contents

1 Summary

A graph is an intuitive abstraction that naturally captures data entities as well as the relationships among those entities. It embeds complicated entity relationships more succinctly, compared with the tabular representation in relational databases. With the power of intuition and succinctness, the graph representations are adapted into a wide spectrum of domains.

Thanks to the advantage of graph representations, researchers have employed graph representation in advanced domains like bioinformatics and social network study. Complications arise sometimes from sheer size of entities, sometimes due to varieties of relations. Discovering the underlying relationships becomes a more demanding task. This task requires not only identifying critical information (graph

patterns), but also presenting it intuitively. The process of pattern identification is termed as **graph mining**, while presenting it in a graphical form is defined as graph visualization.

There is one class of critical information within a graph that catches most research attention, and it is called the dense subgraphs. A dense subgraph (pattern) is one class of critical information within a graph that represents a high level of interactions among entities. Such high level of interactions in many applications implies outstanding level of interactions. It catches most research attention and is also the focus of this thesis. It addresses the computational difficulty, the interpretability and the results' availability during the mining process of dense graphs.

Our thesis is organized in the following way. Firstly Chapter 4 introduces an algorithm called CSV, that mines dense patterns (a.k.a. cohesive subgraphs) effectively. Besides discovering cohesive subgraphs, it also produces an ordering of the vertices for further visualizing of the mining results. As CSV needs to detect cliques (a fully connected pattern) within the graph, which runs in exponential time, we propose a technique to reduce the algorithm's running time. The technique swiftly computes an upper bound on the size of cliques within the graph instead of trying to determine the exact clique size. By this means, we reduce the running time significantly compared with a state-of-the-arts dense pattern mining algorithm CLAN[WZZ06], based on experiments performed on real datasets.

Although CSV performs significantly better than CLAN[WZZ06], in the worst case, it still exhibits high running time. In Chapter 5, we employed triangle counting in dense subgraph mining, which enables us to handle large graphs more efficiently. In this chapter, we propose a set of triangulation (the process of counting triangles inside a graph) based solutions to mine DN-graphs [1] from large graphs. This set of solutions target at different dense pattern mining settings, ranging from in-memory to disc based graphs, and from static to dynamics. Experimental study shows that it is able to produce high quality results within one hour for world-wide photo sharing network Flickr [Inc10].

In Chapter 6, we showcase the **DVIG**, an on-demand visualization system for graph mining pattern. DVIG presents the dynamic patterns in an intuitive manner so that users can capture major trends of the target graph over time. Technical contributions include an intuitive summarization of discovered graph patterns.

With above work, we conclude the thesis in Chapter 7 and discuss future work.

[1] Intuitively, the DN-graphs are sub-graphs share more neighbors than its surroundings, Chapter 5 will cover it in detail.

List of Figures

List of Tables

2 Introduction

A graph captures data entities and their relationships in an intuitive manner. Data entities are represented as vertices in the graph and edges capture the binary relationships between vertices. Although on-going researches in different domains may create the need to capture non-binary relationships , we can still use several graphs to decompose those non-binary relationships into binary ones (similar cases occur in traditional DBMS, where non-binary relationships are resolved via table joins).

Graph representations are flexible and can be used to model data from a number of domains. In financial market monitoring, the graph model captures the correlations of stock prices, which analysts use to infer stocks' fluctuation in future.

In molecular biology, protein protein interactions, which are the most fundamental activities in any living cell, are modeled as graphs. While in social relation analysis, the interpersonal relationships are best abstracted as social networks. The reason behind common choice of graphs is that data's graph representation can picture the changing trends more succinct and more intuitively, compared with non-structured ones.

In this introduction, we firstly present examples of real life graph patterns and their implications in several application domains in section 2.1 . A pattern indicates a group of entities that behaves similarly, be it a group of correlated stocks or a set of homogenous proteins. After that, section 2.2 further reviews challenges in dense subgraph mining. We articulate the problems we would like to solve, give an outline of works we have accomplished, and highlight the contributions of this thesis in section 2.3. With an outline of the overall thesis, we end this chapter.

2.1 Phenomenon of Graph Patterns

Over the past few years, we have witnessed the growing popularity of graph representations in various domains. With technology advancing, the application domains of graphs become more complicated - more entities emerge and more interactions. This brings challenges when we try to extract graph patterns. As the graph keeps expanding, the pattern search space grows exponentially. In a massive graph, we cannot afford to verify each candidate when searching for patterns, even when it is static, not to mention more complicated situations when the graph topology evolves over time. Taking one step back, when the graph is extremely large, even collecting statistics to analyze its topological properties is difficult. We thus have no better way to locate high information content sub areas, except searching for sub areas that are substantially different from the rest of the graph.

Dense graph patterns (characterized by outstanding number of edges embedded) are semantically prominent in many application domains, such as:

- Stock Market Analysis

  The primary task in stock analysis is to predict stocks' price change for tasks such as estimating future return, allocating portfolio and controlling risks. The stock price correlation graph contributes greatly in the analytical process. Normally, the correlation history is transformed into a correlation graph. Graph vertices and edges indicate stocks and their prices correlations respectively. The dense patterns inside the correlation graph are typically groups of companies involved in related industries or having implicit connections in between. E.g. in the study carried out by [BBP06], researchers discovered a dense pattern consisting of companies from IT industry sector, such as Sun Microsystems Inc., Cisco Systems and IBM etc. The expansion of the patterns over time indicates the booming of IT industry in the 90's.

By interpreting the stock correlation, financial professionals can observe industry development trend to make wise investment decisions.

- Protein Protein Interaction

  Biologists also observe the phenomenon that dense graph patterns exist in protein protein interaction process. The protein protein interactions are the fundamental activities for numerous living cells. The interactions among proteins are represented as a graph. The vertices are individual proteins and two proteins have an edge if they participate in some biological process. Researchers discover that a dense graph pattern inside the protein protein interaction graph often indicates that these proteins have similar behavior. This knowledge may further facilitate functional annotation[HYH$^+$05, AUS07].

- Social Network

  In the domain of social relations, dense patterns disclose critical information such as community structure. A social network is a graph whose vertices represent people and an edge connects two vertices if two people have certain relationships. The Digital Bibliography & Library Project (DBLP) network is an instance of a social network to capture the academic publication community in computer science. DBLP records relations such as co-authorship and article-reference for further citation and referencing purposes. The dense patterns in DBLP graph represent research groups, or highly relevant papers.

Graph patterns especially dense patterns have various implications in wide range of application domains. Researchers thus strive to seek for efficient solutions for locating these patterns. The problem of mining (dense) graph patterns becomes center of many research projects ([ARS02, AUS07, BBP06, BC96]). With much effort put into the dense pattern mining research, researchers have realized that finding dense pattern is a challenging task.

2.2 Dense Pattern Mining's Challenges and Our Solutions

Graph representation is more succinct when capturing complex relationships compared to tabular representation. This compactness however, comes at a price. When mining dense patterns, the fundamental task of deciding whether a subgraph is a dense pattern becomes difficult, as it is closely related to well-known NP-complete clique detection problem. Subsequently, it takes intolerable long time to mine huge graphs. What's worse, even after locating dense patterns, we can hardly interpret the semantics of the patterns due to its structural complexity. In the following parts, we explain above mentioned challenges and our solutions in detail.

- It is computationally expensive to identify dense patterns

  The primary question for dense pattern mining is to decide whether a subgraph is a dense pattern. To answer this question, an algorithm needs to check the candidate's internal connections. For a dense pattern, the connections are outstandingly intensive with respect to its neighbors. Here the neighbors are the pattern's immediately connected vertices. When searching for dense patterns, existing algorithms enumerate possible vertex candidate sets. This enumeration results in combinatorial algorithmic complexity. When the pattern is a fully connected subgraph (or in graph theory terms, a clique), the algorithm is detecting cliques. In [**?**], Karp proves that identifying dense patterns (which are almost cliques) requires combinatorial complexity.

  Facing the complexity challenge, we opt for estimation with highly accurate results, meanwhile overcoming the combinatorial complexity. One feasible proposal is to provide an density upper bound for dense patterns within the limit of computational resources. This upper bound can subsequently reduce search space when requests to exactly locate dense patterns arise. Chapter 4 presents our method of calculating the upper bound and explains how to detect dense patterns with the upper bound.

- Large-scaled graphs processing faces physical constraints

When the graph size keeps on growing, it is even harder to locate dense patterns. In extremely large graphs, simple tasks, such as loading graph links, is challenging, not to mention more complicated tasks. The mining of dense patterns depends on atom operations such as graph link scans. These fundamental tasks consume extraordinary computational resources and s-torage space. For instance, WWW has already reached 10^{10} indexed web pages in year 2005, and each of the pages typically has twenty to thirty links [GS05]. Even the world-leading search engine provider, Google, strives to cache every web page and scan them periodically for updates. This simple routine has already cost Google enormous energy and resources. Imagine the resources needed when carrying out mining operations on WWW. This calls for breakthrough in efficient mining of huge graphs.

In Chapter 5, we propose a triangulation-based solution to efficient mining of huge graphs. This approach has three advantages. Firstly, most of the de-tails involved in efficient processing like minimizing I/Os etc. are abstracted within the triangulation algorithm. The abstraction ensures our approach's extensibility to different input settings. For example, when the graph to be mine is too large to fit into memory, our approach only needs to change the accessing method of the graph links. The estimation of local neighborhood is enclosed in the triangulation algorithm. Secondly, as the estimation of local density value improves in every iteration, users are able to obtain the most updated results at any instance during the course of algorithm running. Finally, when the graph is too large to fit into main memory, we can collect statistics regarding the graphs in the first iteration to support effective buffer management for storing the local density value on a disk. The collection of statistics can be accomplished since the triangles come in the same order in every iteration.

- Dense graph patterns are hard to be interpreted

  In additional to effective algorithms, the collection of discovered dense patterns need further processing to be meaningful to human beings. A dense pattern embeds domain knowledge into its implicit structure. Its relationship with other patterns also carries functional information. To interpret the information, human analysts need to organize the pattern's internal structure as well as its connections with other patterns. The interpreting process becomes tedious when facing a large volume of mining results.

  To free human beings from tedious works of organizing patterns, we developed a visual system DVIG. DVIG is a lightweight graph mining pattern visualization tool. It assists domain experts in understanding the summarization as well as individual mining graph patterns from external graph mining algorithms. DVIG offers a visualization paradigm for dynamic graph pattern visualization, and provides features to present semantics when visualizing domain data. The detail of DVIG system is explained in Chapter 6.

2.3 Thesis Contribution

With above proposed solutions, this thesis presents three relevant pieces of work for graph dense pattern mining. The contributions of this thesis are summarized below:

2.3.1 Contribution 1: an Algorithm that Locates Dense Subgraphs Effectively

The first work (which appears in Chapter 4) concerns how to locate the dense subgraphs. More specifically:

- We propose a novel algorithm called CSV to compute an ordering on graph vertices. CSV also has the capability of visualizing cohesive (a.k.a. dense) subgraphs within the graph.

- As algorithm CSV needs to detect cliques within the graph, we propose a technique to minimize running time. The technique swiftly computes an upper bound on the size of cliques within the graph instead of deciding exact clique size. By this means,our algorithm is up to 100 times faster compared to a state-of-the-art dense pattern mining algorithm CLAN[WZZ06].

The technique employs a novel mapping to transform graph elements (vertices and edges) into high-dimensional points while preserves graph elements' connectivity relations. After the transformation, existing spatial indices such as the R-tree can be applied to the transformed data. This makes CSV more extendable to handle larger graphs.

- In addition to using CSV as a stand-alone tool for mining of dense subcomponents, we also pre-filter graph data using CSV to significantly speed up exact clique finding algorithms such as CLAN[WZZ06]. Experiments shows that CSV can save up to 84% running time.

2.3.2 Contribution 2: Triangulation-Based Dense Pattern Mining

Subsequently, we propose a triangulation-based solution to further mine larger scaled graphs. We present our research findings in Chapter 5. In this work, we achieve the following:

- We look at dense sub-graphs from a new perspective. A dense subgraph contains a set of highly relevant vertices,which share many common neighbors(two vertices are neighbors if they are connected by an edge). With that in mind, we define the DN-graph, a more general view of dense patterns discussed in Chapter 4. This definition lays foundation for triangulation-based dense pattern mining.

- This work proposes a set of triangulation-based solutions to mine DN-graphs from large graphs. This set of solutions target at different dense pattern mining settings, ranging from in-memory to disc based graphs, from static to dynamic. Experimental study shows that it produces quality results within one hour for world-wide photo sharing network Flickr [Inc10].

2.3.3 Contribution 3: DVIG, a Dynamic Visualization System

In chapter 6, we showcase the **DVIG** dynamic visualization systems for graph mining pattern. DVIG presents the dynamic patterns in an intuitive manner so that users can capture major trends of the target graph over times. Technical contributions include:

- An intuitive summarization of discovered graph patterns. Being an effective visual tool, it is not sufficient to only provide visual image for individual graph patterns. Since the discovered patterns are overwhelmingly numerous, a wiser choice is to profile all interesting patterns and present the meta information first before dill down into a specific graph pattern. Preferably, the meta information should include indicative measurements of patterns' interestingness, and guide users for potentially prominent patterns. Hence domain experts are able to investigate patterns discovered on state-of-the-arts graph mining algorithms while not hindered by the complexness of understanding mechanisms behind these algorithms.

- An layout scheme that organizes the discovered patterns into a force-directed structure. This layout captures the inter and intra relationships among discovered patterns. It also possesses the dynamic power of display the changing trend of the graph patterns discovered due to the evolving of underlaying graphs. The effect of time towards the interactions are better observed and are ready for further analysis.

2.4 Outline of the thesis

The rest of the thesis is organized as follows: Chapter 3 gives a more detailed description of the dense graph patterns, reviews commonly adapted dense patterns and surveys state-of- the-art graph mining and visualization systems built by different institutes and organizations.

Chapter 4 presents our proposed technique for locating dense subgraphs effectively: a cohesive subgraph mining algorithm CSV. The CSV solution consists of three steps. Firstly, it utilizes a special space mapping to transform the graph vertices and edges into high dimensional points. Secondly, it builds spacial index on the transformed points. This index facilitates locating cohesive subgraphs.

Chapter 5 proposes a triangulation-based solution to extend CSV to larger graphs. This work firstly provides an innovative way of using triangle counts to locate dense patterns. Secondly, we extend this solution to handle streaming graphs, whose vertices can fit into main memory, while edges reside in secondary storage media.

As visualization always plays an important role in interpreting graph mining results, we develop a visualization tool for dense pattern mining. Chapter 6 showcases a visualization system DVIG: DVIG is a lightweight graph mining pattern visualization tool. It assists domain experts in understanding individual graph patterns and provides a summary of patterns. It also possesses the capability of visualizing patterns' dynamics from external graph mining algorithms.

With the above work, we conclude the thesis in Chapter 7.

Two papers have been published based on the work presented in this thesis. The work on cohesive subgraph mining algorithm has been published in [WSTT08]. The triangulation based DN-graph mining work is to appear in [WZTT11].

3 Literature on Graph Model and Mining Algorithms

Graphs are the most pervasive model of entity interactions. Compared with unstructured representations, graphs are more concise and intuitive in abstracting entity interactions. When using graph representation, we do not need storage for every combination of binary relationships. Another advantage is that graph requires no normalization of relations which may lose data integrity [DBLEH07].

Inside a graph, some parts have more connections inside. Many researchers agree that these dense patterns capture the most active involvement of entity interactions thus prominent [ABC$^+$04, ARS02, ATH03, BBP06, HYH$^+$05]. The application of dense patterns can be found in various domains. In domains of social network, a dense pattern indicates community. While in protein protein interaction networks, a dense pattern may tell us functional similarity among proteins [HYH$^+$05].

Graph mining is a special category of structured data mining. The process of graph mining is to abstract useful information from graph data, be it a collection of graphs or a huge graph. In addition to getting useful information, it is more desirable to present the findings in an intuitive way. This aids in better understanding of the semantics of the findings. With proper manifest, the distribution of the patterns is also revealed.

The rest of this chapter is organized as follows. We first introduce current understanding about graph dense patterns (section 3.2). After that, we investigate related research works on effectively discovering of dense patterns. As visualization is an important aspect in interpreting the graph patterns, this chapter then discusses recent effort in visually presenting graph patterns.

3.1 Graph Data Model

A graph is a collection of items and their relationships. The items are graph vertices, while their relationships are graph edges connecting two relevant vertices. If the relationships are associative, we use undirect graph to model it. If the relationships are comparable among themselves, we usually assign weights to the edge. The weights are quantitative measures which enable relationship comparison. Through out this thesis, we concern only undirect un-weighted graphs. Other classes of graphs can be transformed into this primitive model of graphs via setting thresholds.

3.2 Dense Graph Patterns

A dense graph pattern is a connected subgraph that has significantly internal connections with respect to the surrounding vertices. Depending on the semantic meaning of the graph data, various forms of dense patterns are investigated in literature.

- **Clique/Quasi-Clique**

  A clique represents the highest level of internal interactions. Originally, the meaning of the word clique is an inclusive group of people who share interests, views, purposes, patterns of behavior, or ethnicity[Sco00]. In graph theory, a clique is a fully connected subgraph. Each pair of vertices are connected by an edge. While a quasi-clique is an "almost" clique with few missing edges. If a clique is not a proper subgraph of a larger clique, we call it a "closed clique".

  Recent researches confirmed that closed cliques/quasi - clique have important domain implications [ABC$^+$04, ARS02, ATH03, BBP06, HYH$^+$05]. The clique related patterns in cellular phone calling networks indicate families, project teams or complicated romance relationships [ARS02]. While closed cliques and quasi - clique in scientific network such as protein protein interaction networks indicates potential protein complexes[HYH$^+$05].

- **High Degree Patterns**

  Another kind of dense substructures are high degree patterns. Inside them, the average vertex degrees are above certain level or are outstanding among surrounding vertices. Here a vertex's degree refers to the number of edges intercepting the vertex. Different from clique relation patterns, the high degree patterns do not require high interconnection within the pattern. As long as the vertices in the pattern have high degree in the graph to be mined, it is included into the pattern ([GRT05] targets at these patterns). Thus solution of mining high degree patterns only need to compute every vertex's degrees once and ensure the discovered patterns are connected subgraphs.

- **Dense Bipartite Patterns**

  If the entities involved belong to two classes, and only entities from different classes have associations, the graph is a bipartite graph. Similarity, a dense bipartite pattern is a bipartite graph with outstandingly many edges.

  The dense bipartite patterns arise in social network domain. In social networks, we constantly discover the structure of hubs and authorities. Many researches argue that they are the core of communities [RRRT99]. In order to search for the "signature" of communities, they look for a dense bipartite graphs.

- **Heavy Patterns**

  Previous patterns emphasize on the topological features. The heavy pattern, on another hand, focus on the maximality of edge weights. The research in [SK98] calls subgraphs of fixed number of vertices a heaviest pattern if the sum of edge weight are maximized. The heavy pattern has closed relationship with this paper's focus: dense pattern. If graph edges' weight follows triangle inequality, the heavy pattern is also a dense pattern in the un-weighted graph.

  Even though this type of pattern is not our preliminary target for this thesis, it is presented here for completeness.

3.3 Background of Graph Mining

Graph mining emerges along with the explosion of structured data. Advances in technology have enabled us to collect vast amount of structured data across a myriad of domains for various purposes, ranging from computational simulations to network flow data, from genomic data to web access and linkage statistics. Different from unstructured data, these data have complicated relationships within. Depending on the level of structure imposed, the structured data are extended from semi-structured ones such as XML to well-structured form (for example, ordered tree). The enormous amount of structured data require efficient graph mining tools to abstract useful patterns out.

The application of the graph mining in computer science domain are heterogenous from database applications to machine learning area[HK00]. In data base applications, graph mining discovers structured patterns from multi-relational data base[Der03].In machine learning area, graph mining problems are approached via kernel function centric, SVM-based methods[SMT91]. Without graph mining techniques, the task of locating patterns requires logical analysis and domain experience.

This thesis surveys graph mining research and classifies them into two directions. One direction is extending classical data mining concepts to graphs. This direction targets at discovering context-free patterns. The other direction is taking into the account of domain knowledge. Before presenting research literature from the two directions, we first discuss current research advance of graph matching research, which is a fundamental challenge many graph mining techniques face.

3.3.1 Basic Problem: Graph Matching

The matching of subgraphs is the performance bottleneck particularly in the Apriori-based graph mining algorithms [ATH03, MK01, YH02, MK01]. Apriori refers to a search paradigm that searches in breadth-first manner and uses a tree structure to count candidate subgraph sets efficiently. It generates candidate subgraphs of size k from size $k-1$. Then it prunes the subgraphs which does not satisfy mining criteria [AS94]. According to the downward closure lemma, the candidate of size $k+1$ can not contain non-candidates of size k. During the candidate generating process, we use graph matching to decide whether the two candidate subgraphs are the same.

The process of graph matching is to form a one to one vertex mapping from one subgraph to another subgraph, such that mapped vertices have the same topological structure. Graph matching is one of the complicated problems in graph theory domain[Bas94, GJ79]. In fact, graph matching is NP-complete [GJ79].

The graph matching problem is: Given a model graph $G_M = (V_M, E_M)$ and an input graph $G_D = (V_D, E_D)$ with $|V_M| = |V_D|$, look for a one-one mapping $f : V_D \to V_M$ such that $(u, v) \in E_D$ if and only if $(f(u), f(v)) \in E_M$. If a mapping exists, we call f an isomorphism from G_D to G_M. Searching for this mapping is the problem of exact graph matching. If it is not possible find such mapping f, for example, the number of vertices are different in the 2 graphs, the problem becomes looking for best matching between them. In that case, it is the inexact matching problem. Figure 3.1 gives an overview of the classification of the Graph matching problems.

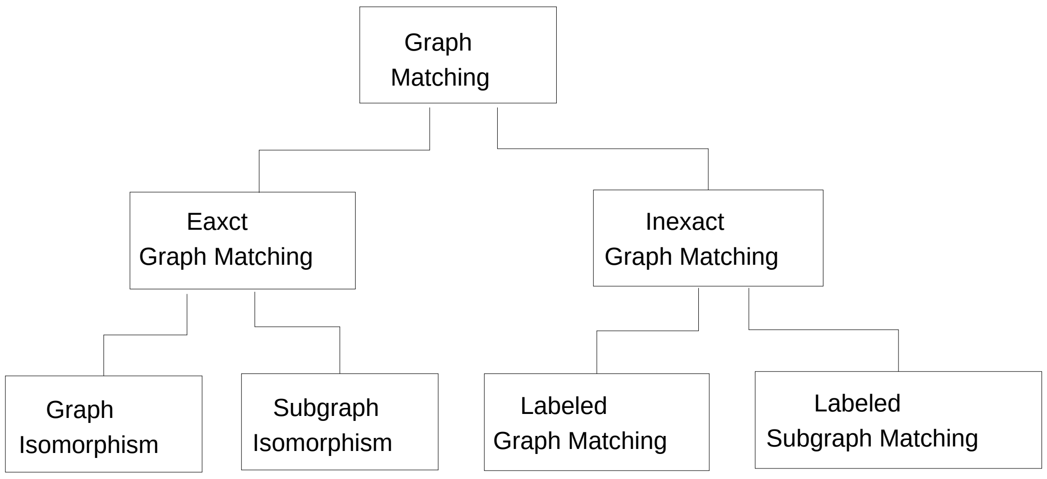


Fig. 3.1: Graph Matching Problem Classification

3.3.1.1 Exact Matching

The exact matching problem hasn't been classified into any type of complexity such as P or NP-complete yet. Depending on different assumptions on graphs, [GJ79] and [Bas94] prove it as an NP-complete problem. Other researchers have found polynomial solutions for some special graph classes such as planar graphs [HW74]. Generally speaking, the complexity of the whole problem class remains as an interesting open theoretical problem.

There are two categories of exact matching algorithms. The first approach is based on group theory. It classifies the adjacent matrices into permutation groups.

The second approach constructively forms graph isomorphism.

Group Theory and Graph Matching [Bas94] gives a moderately exponential bound for the general graph matching problem. If the graphs have constraints, the matching problem is possible to have a polynomial bound[Luk82]. However, the above approaches is only of theoretical interest due to its large constant overhead.

Practical Graph Matching

Depth-first backtracking search is the most established practical algorithm class for graph matching way back in 70s. Based on that, further improvements, such as combining backtracking with a forward checking procedure, are develope-d. The basic idea for forward checking [Ull76] is: For an established matching, check whether there is at least one mapping when adding more vertices. By this way, an algorithm can immediately reject the mappings with no further extension. Search space is thus reduced significantly.

Clique Searching is a class making use of association graphs and clique searching [MARW90, KH04]. Inside an association graph, each consistent pair of vertices, which are eligible to form a mapping, form an association vertex. An association edge links two locally consistent mapping pairs. By this way, the maximal clique in the association graph represents the largest common subgraph between the two original graphs. The matching problem transforms to maximal clique finding problem.

Rooted from clique detection, this approach has the drawback of high computational complexity. Given two graphs G_M and G_I with n and m nodes respectively, the size of the association graphs and the number of possible cliques strongly depend on the number of labels in G_M and G_I. The association graphs increases its edge exponentially with the increase of consistent vertices.

Decomposition approach [MB00] tackles graph matching problem by firstly decomposing the input graphs into smaller subgraphs. Next, it matches these small pieces with the model graph respectively. The efficiency of such approach depends on the choice of decomposition policy.

There are many different decompositions for a given graph. Find the optimal one is expensive. It is more efficient to look for some sub-optimal yet inexpensive ones. This approach is most suitable for relational database applications. However, it still faces the problem of exponentially increase of decomposition choices. As for the problem of graph mining, since we are not sure about the target sub-pattern's distribution before performing the operation, it is possible that we separate graph patterns into different pieces during decomposition process.

Computational complexity of practical exact graph matchings To summarize above mentioned graph isomorphism checking technologies, we use table 3.1 to list theoretical computational bound for respective technologies. The bounds are presented using the following quantities:

- D = number of input graphs

- v_{dc} = number of vertices of a subgraph that is common to all input graphs
- v_{du} = number of vertices of a subgraph that is unique to each input graph
- v_d = total number of vertices of a input graph
- v_i = the number of vertices of model graph

Algorithm	Heterogeneous Database		Identical Database	
	Best	Worst	Best	Worst
Clique Detection	$O(Dv_dv_i)$	$O(D(v_dv_i)^{v_d})$	$O(Dv_dv_i)$	$O(D(v_dv_i)^{v_d})$
DF Backtracking	$O(Dv_dv_i)$	$O(D(v_d^2v_i^{v_d}))$	$O(Dv_dv_i)$	$O(D(v_d^2v_i^{v_d}))$
Decomposition	$O(Dv_dv_i)$	$O(D(v_d^2v_i^{v_d}))$	$O(v_dv_i)$	$O(v_d^2v_i^{v_d})$

Tab. 3.1: Complexity Comparison among Different Exact Matching Algorithms

Table 3.1 shows that Depth-First backtracking and decomposition approaches out-perform clique detection based algorithm in the worst case. Yet if the database has more homogeneous graphs, decomposition based approach will be more efficient than the Depth-First backtracking.

exact graph matching	unlabeled	labeled
<20 vertices	decision tree method DF-backtracking	decision decomposition
<500 vertices	DF-backtracking	decomposition
$\geq$ 500 vertices	continuous optimization methods	

Tab. 3.2: Suitability of Exact Graph Matching Paradigms [MB00]

Table 3.2 from [MB00] suggests how to make decisions on graph matching methods when facing different graph classes.The decision is based on graph characteristics such as the size of database graph and the number of labels appearing in the graphs.

In summary, most above mentioned exact matching algorithms face high computational complexity. This restraints their application for large graphs. In fact, in practical applications, we may not necessarily do exact matching. A high quality approximate match usually is a wiser choice.

3.3.1.2 Inexact Matching

Compared with exact matching counterpart, inexact matching solutions are more cost effective. For example, in computer vision and pattern recognition, graph models are commonly used to represent fuzzy objects such as Chinese characters and hand-draw images. These objects carry noise. It is thus desirable to look for error-correction (inexact) graph matching methods. Inexact graph matching algorithms commonly adopt heuristics, such as genetic algorithms and simulating annealing, to improve matching efficiency while tolerant errors. It is thus not surprise to see that many graph matching research favors this types of solutions.

Inexact solutions of graph matching commonly require measurements to define the similarity between two graphs. Besides calculating the similarity, a solution also needs a threshold to filter out unmatched subgraphs.

The most well-adapted metrics of graph similarity is graph edit distance. Similar to string edit distance, the edit distance between two graphs is the minimum cost of a sequence of edit operations that change one graph to an isomorphic graph of another. [Bla94] gives a formal definition of the graph edit operation, for labeled graphs. A labeled graph is a graph G with 4-tuple $G = (V, E, \mu, \nu)$, where

- V is the set of vertices.
- $E \subseteq V \times V$ is the set of edges.
- $\mu : V \rightarrow L_V$ is a function assigning labels to the vertices.
- $\nu : E \rightarrow L_E$ is a function assigning labels to the edges.

The edit operations δ on labeled graph are defined as any of of the following:

Definition 1. *[Bla94]*

- *vertex label substitution: substitute the label $\mu(v)$ of vertex v with another label l.*
- *edge label substitution: substitute the label $\nu(e)$ of edges e with another label l'.*
- *vertex deletion: delete the vertex v and all edges connected to it from the graph.*
- *edge deletion: delete an edge e from the graph.*
- *vertex insertion: insert a vertex v into the graph.*

- *edge insertion. insert an edge e between 2 existing vertices in the graph.*

Based on this set of edit operations, the error-correction subgraph isomorphism (a.k.a. inexact graph matching) is defined as following:

Definition 2. *[Bla94] Given a labeled graph $G = (V, E, \mu, \nu)$ and a sequence of graph edit operations $\Delta = (\delta_1, \delta_1, ..., \delta_k)$, the edited graph $\Delta(G)$ is a graph $\Delta(G) = \delta_k(...\delta_2(\delta_1(G))...)$. The error-correcting (ec) subgraph isomorphism f from G to G' is a 2-tuple $f = (\Delta, f_\Delta)$ where*

1. *Δ is a sequence of edit operations such that there exists a subgraph isomorphism from $\Delta(G)$ to G'*
2. *f_Δ is a subgraph isomorphism from $\Delta(G)$ to G'*

The graph edit distance between two graphs is defined as:

Definition 3. *For two graphs G and G', the graph edit distance from G to G', $d(G, G')$ is the minimum cost over all error-correcting subgraphs isomorphisms f from G to G': $d(G, G') = \min_\Delta\{C(\Delta)|$ there is an ec subgraph isomorphisms f from G to G' $\}$.*

Computing the exact graph edit distance is not an easy task. The combinatorial nature of this problem makes the heuristic the only remedy.

3.3.2 Recent Advances in Graph Mining

Recently, research on discovering context-free graph patterns such as frequent subgraphs and closed graph patterns become more and more active, due to the demand for knowledge abstraction from increasingly large volume of graph data.

The researches on mining frequent subgraphs fall into four categories [WM03]: Greedy methods, Apriori-based approaches, inductive logic programming (ILP) oriented solutions, and methods based on feature selection.

Greedy search based approaches emerge along the research on discovering conceptual graph inside a graph. SUBDUE [HCD94] and GBI [YMI94] are two pioneer algorithms using this approach. They both greedily look for a subgraph that maximizes certain measurements. After that, they compress the discovered subgraph into a vertex. SUBDUE adopts Minimum Description Length principle and GBI uses an empirical graph size definition. This definition considers both the size of the discovered subgraph and the size of the compressed graph.

Apriori-based approach is the most explored branch. Solutions under this approach often represent a graph using some canonical forms. The canonical forms impose a partial order to graphs vertices or edges. Based on the partial order, the solutions explore the graph in the Apriori-based manner. AGM [ATH03] and F-SG [MK01] are two well-known Apriori-based frequent sub-graph discovering algorithms. They both adopt the breadth-first Apriori algorithm. In contrast, gSpan[YH02] tries to apply Apriori principle to a graph labeling in depth first manner.

Solutions based on ILP represent a general graph by first order predicate logic. Graph mining problem is transformed to looking for hypothesis and using ”evidence” to justify the hypothesis. The hypothesis is constrained by background knowledge. The background knowledge thus have influence on the types of mined subgraphs. System WARMR[DT99] and its improved version FARMER[NK99] are the first few among ILP based graph mining approaches. By combining ILP approach with Apriori-based search, they can predict carcinogenesis of chemical compounds.

The feature selection approach is an application of SVM[Vap95] to graph mining domain. The key issue is to find a suitable kernel function to measure the similarity between two patterns [KI02]. Based on the kernel functions, patterns are classified properly. We can find target patterns by analyzing most representative patterns from each class.

Along with the discovery of more functional-structural correspondence, researchers in bio-informatics domain discover that meaningful patterns usually inherit certain structural characteristics. Recently a lot of research papers have been published on cliques, densely-connected graph and another interesting graph class the bipartite graph[ZWZK06, AUS07, WZZ06, HYH$^+$05, YZH05, NJW01, KV96, HMWD04]. Below we briefly describe the work that is most relevant to the dense pattern mining problem.

One approach to mining dense patterns is via graph partitioning along a minimum cut. Ng and colleagues [NJW01] use spectral methods to interrogate large complex networks in order to find cohesive (a.k.a dense) subgraphs. Since the eigenvectors of a graph depict the graph's topological information, they perform spectral graph partitioning using the second eigenvector of a graph's Laplacian (also known as the Fiedler Vector). The algorithm recursively bisects the graph until the number of partitions meet predefined values.

METIS [KV96] is an algorithm for graph partitioning, a related but different problem to the theme of this thesis. The algorithm comprises three stages: a coarsening step where tight groups of vertices are replaced with super nodes; a partitioning step where the resulting coarsened graph is partitioned; and a refining step where the original vertices are added back to the graph and the partitions are refined accordingly. The approach is scalable but METIS targets a different task – graph partitioning, and moreover tends to favor balanced partitions and does not always effectively identify dense sub-graphs of interest[AUS07].

By applying special encoding techniques, some solutions convert the dense graph mining problem into a traditional data mining algorithm. The Apriori priciple[AS94], can be directly applied[ATH03, MK01, YH02, HYH^{+}05]. Out of many attempts, the one that is most germane to our work is CODENSE[HYH^{+}05]. CODENSE mines frequent coherent cohesive subgraphs across a collection of massive graphs. A coherent subgraph is a graph where all edges exhibit correlated occurrences in the collection of graphs. The density of a graph is the ratio of graph edge count in the graph to the same-sized, complete graph's edge count. CODENSE performs clustering on two meta-graphs summarizing the graphs to be mined. CODENSE can discover overlapping clusters. This is important in biological applications since under different conditions, one gene may serve different roles and be involved in different functional groups [GE02]. Aside from density, in [YZH05], studies are also done on efficiently finding interesting subgraphs through combining connectivity constraints with column or row enumeration methods [PCT$^+$03, CTX$^+$04, CTTP04, PTCX04, CTTX05].

In summary, the request for an efficient dense subgraph mining algorithm urges us research on this topics. If we can derive an approach that can visualize mining results, users can better identify graph structural features concretely. We next describe effort researchers made to provide such visual graph mining capabilities.

3.4 Visualization of Mined Graphs

Mining result visualization(MRV) is an important step towards a human-interpretable data mining solution. With the aid of effective visualization techniques, analysts can interpret and explore the mining results intuitively. Moreover, the human feedback upon the visual representation may positively influence the mining algorithm since it builds up an intuitive mental model of how the mining algorithm works.

As with its counterpart in unstructured mining, the development of graph mining results visualization can also be classes into two stages. Early works emphasis on displaying findings on the graphs. When mining huge graphs, it is natural to organize the resulting subgraphs into hierarchy so that humans without prior knowledge of graph mining technology can understand the meaning of discovered patterns easily. Current state- of - the -arts [CFZ06] in graph mining result visualization adopts such approach. Further development in this area makes interactive exploration of the graph mining results[RJTe06].

3.4.0.1 Interactive Graph Mining Tools

Interaction with users during mining process effectively guides further mining process. It provides users with a more intuitive mental model. To our knowledge, few researches focus on interactive features. Most of them are capable of visualizing the graph topologies in different magnitude. They cannot presents intermediate mining results for users to feedback and improve. **Netmine**[CFZ06] and **GMine**[RJTe06] are two systems that provides interaction during mining process.

Netmine[CFZ06] is a systems to visualize network properties such as minimum-cut. It includes an A-plots tool to present adjacency matrix of a graph visually to facilitate outlier detecting. It also contains a procedural based R-MAT (Recursive MAtrix) graph generator which produces graph patterns following power law. By choosing proper parameters, the R-MAT graph generator produces a synthetic graph that matches with the real data well. The failure of matching indicates network abnormalities fast and accurately. To generate a bipartite graph, the adjacency matrix is rectangle. By setting the length and width of the adjacency matrix to be different power of 2, it matches bipartite Click-stream data sets well.

GMine[RJTe06] is an initiative to perform interactive graph mining. It does not only provide general topological information of the graph but also helps users to look into of the detail of the graph. By hierarchically partitioning and arranging the graph, it enables exploration of the graph in different scales. It proposes an R-tree-like structure G-tree to store the graph to enable multi-resolution graph exploration. Depending on the choice of partition methods, GMine is capable of displaying the graph in different ways according to different application domains' specific requirements. However, one serious limitation of this system is its failure to deal with overlapping subgraph patterns due to its partition nature. Another contribution is an interesting measure of the subgraphs. The score is based on steady-meeting probability that two random moving objects meet each other at the given node. More intuitively, it is the closeness of the connections among vertices inside a subgraph. To locate the critical paths, GMine uses dynamic programming. This proposed score has the tendency to group densely-connected vertices into a single G-tree node. GMine is suitable for isolating dense areas while may not be able to handle overlapping patterns.

Although existing works make graph mining a navigational process, there is no work to our knowledge that can handle adding and deleting vertices or edges during navigation of the mining results. This missing piece towards the solution of interactive graph mining worth time and energy to be investigated. Taking protein protein interaction network as an example, the quantities of interactions are from experimental data. Due to current experimental constraint, these data contain high level of noises. Thus the interactions inside the protein-protein network may not be 100% accurate. To overcome noisy data, we can adjust under lying network by mining results and iterative such process to get optimal results. From this iterative process, we may predict missing links in graphs or discover critical vertices and edges.

3.5 Chapter Summary

In this chapter, we first discuss the graph data model to capture entity interaction-s. After that, we present several important classes of dense graph patterns in the graph mining literature. With different characteristics, these patterns have het-erogenous implications in various application domains. Next, we investigate the evolvement of techniques for graph mining with emphasis on dense patterns. Since graph matching is an elementary problem during mining process, we review the graph matching techniques as part of the evolvement. After that, we extensively survey mining techniques for afore mentioned patterns. In the last part of this chapter, with the discussion on current trend in graph mining visualization, we reconcile that visualization plays an important role in graph mining.

4

Cohesive Subgraph Mining

Cohesive subgraph is a synonym of dense subgraphs. Inside a cohesive subgraph, vertices connect to each other with significantly higher number of edges. Cohesive subgraphs have significant implications in various domains as discussed in section 2.1. The answer to the questions of where cohesive subgraphs are, however requires excess effort.

This chapter presents our solution towards cohesive subgraph mining. In section 4.1, we formally define the cohesive subgraph mining problem. The algorithm to cohesive subgraph mining is introduced in section 4.2. In section 4.3, we conduct extensive experiments on synthetic as well as real data to show the effectiveness of our proposal. Section 4.4 concludes this work.

4.1 Preliminaries and Problem Definition

A graph G is a two tuple $\{V, E\}$ where V is a set of distinct vertices $\{v_1, ..., v_{|V|}\}$ and E is a set of edges $\{e_1, ..., e_{|E|}\}$. A graph $G' = \{V', E'\}$ is a subgraph of G if $V' \subseteq V$, $E' \subseteq E$ and all end-vertices of edges in E' are in V'.

In this paper, we define the distance between two vertices v_i and v_j as the number of edges on the shortest path connecting v_i and v_j. To simplify discussion, we assume G is a connected graph. If G consists of a set of unconnected components, the visualization technique in this chapter can be applied to each component individually.

To determine the cohesiveness of a subgraph, two measurements are often used: density [ARS02, HYH$^+$05] and connectivity [YZH05].

Definition 4. *Density of G', $\gamma(G')$*

The density of a graph $G' = \{V', E'\}$ is defined as:

$$\gamma(G') = \frac{2 * |E'|}{|V'| * (|V'| - 1)}$$

Definition 5. *Connectivity of G',$\kappa(G')$*

The connectivity of a graph G' is defined as $\kappa(G') = k$, where k is the largest integer such that there exists no $k - 1$ vertices whose removal disconnect the graph. We also call G' a k-connected graph.

Intuitively, inside a k-connected graph, there are at least k independent paths from any vertex to any other vertex. The exact algorithm to decide vertex connectivity is not trivial([CT96] proves that this problem is #P-complete).

Although the above two measurements of cohesiveness are inherently different, both are however maximized when the graph is a clique. In this chapter, we will mostly focus on finding this special type of highly cohesive subgraphs (i.e., cliques).

Given a graph $G = \{V, E\}$, we want to compute an ordering of vertices in G and generate a density plot based on the ordering such that:

Problem 1: All cliques with size greater than k can be identified based on visualizing the density plot.

While our focus in this paper is on solving Problem 1, we will provide explanation during the course of discussion on how our solution to Problem 1 can in fact provide a solution to Problem 2 and 3 below:

Problem 2: All subgraphs with density greater than γ_{min} and graph size greater than $size_{min}$ can be identified based on visualizing the density plot.

Problem 3: All subgraphs with connectivity greater than k can be identified based on visualizing the density plot.

4.2 Algorithm CSV

Our CSV algorithm is an OPTICS [ABKS99] style plot which walks through the vertices of the graph based on two local density measures: **maximum participated clique size** and **maximum co-clique size**.

Definition 6. *maximum participated clique size, $\zeta_{max}(v)$*

The maximum participated clique size of a vertex v, denoted as $\zeta_{max}(v)$ is the size of the biggest complete subgraph(i.e., a clique) in G that includes v.

Intuitively, $\zeta_{max}(v)$ is analogue to the notion of spatial density in density-based clustering [EKSX96, ABKS99]. Unlike spatial density, graph density must take into account the number of both vertices and edges. The definition of $\zeta_{max}(v)$ naturally takes this into account.

Given a vertex v_i, we also want to estimate how densely it is connected to another vertex v_j. For this purpose, we introduce another definition: maximum co-clique size.

Definition 7. *maximum co-clique size, $\eta_{max}(v_i, v_j)$*

The maximum co-cliques size for two vertices v_i and v_j denoted as $\eta_{max}(v_i, v_j)$ is the size of the biggest complete subgraph(i.e., a clique) that contains both v_i and v_j.

The CSV algorithm is shown in algorithm 4.1. The algorithm maintains a heap

Algorithm 4.1 CSV Algorithm Skeleton

```
Require: Graph $G$={$V, E$}
  return Density plot for visualizing cohesive subgraphs
  HEAP=$\emptyset$
  for all $v \in V$ do
    v.$\eta_{mseen} = 0$
  end for
  while $\exists v$ unvisited do
    if HEAP==$\emptyset$ then
      $v$=randomly selected unvisited vertex from $V$
    else
      $v$=next vertex on HEAP
    end if
    plot v.$\eta_{mseen}$
    for all $v'$ directly connected to $v$ do
      if $v' \in$ HEAP then
        $v'.\eta_{mseen}$ = max($v'.\eta_{mseen}$, $\eta_{max}(v, v')$)
        reorder HEAP
      else
        compute $\zeta_{max}(v')$ (Algorith 4.2)
        $v'.\eta_{mseen}$=$\eta_{max}(v, v')$
        add $v'$ into HEAP
      end if
    end for
  end while
```

for storing visited vertices that have not been output. Vertices stored in the heap are maintained in sorted order based on η_{mseen} and then by ζ_{max}. The variable $v.\eta_{mseen}$ maintains the highest $\eta_{max}(v, v')$ value between v and any visited vertex v' known so far. Intuitively, sorting the vertices based on η_{mseen} means that vertices that are strongly connected to previously visited vertices will be output first. Among those that have the same value for η_{mseen}, vertices with higher value of ζ_{max} are ranked in front as they are more likely to lead the walk towards the region with higher graph connectivity.

The algorithm starts by either picking the next vertex v from the heap or a random vertex v if the heap is empty. The value $v.\eta_{mseen}$ (which have a value of 0 if it is the starting vertex) will then be output. All vertices v' that are directly connected to v are retrieved. For vertex v' which is already in the heap, $v'.\eta_{mseen}$ is updated depending on whether the original value of $v'.\eta_{mseen}$ or $\eta_{max}(v, v')$ is higher. The heap is then reordered based on the updated value. For a vertex v' which has not been visited, $v'.\eta_{mseen}$ is set to $\eta_{max}(v, v'))$ and $\zeta_{max}(v')$ is computed. After that, v' is inserted into the heap. The process is repeated until all vertices have been visited and its $v.\eta_{mseen}$ has been output.

To see how the density plot output by CSV (henceforth referred to as the CSV plot) can be used to visualize the distribution of cliques, we will first prove the follow theorem.

Theorem 4.2.1. *Let v'_1,...,v'_k be the vertices of a size k clique in G and assume that they are already sorted based on the ordering computed by CSV. We claim that for any vertex v (not necessary those in the clique) that fall between v'_1 and v'_k (excluding v'_1 but including v'_k), $v.\eta_{mseen} \geq k$.*

The proof of theorem 4.2.1 is as follows.

Proof. : Any vertex v that falls within v'_1 and v'_k is either part of the size k clique or it is not.

If v is part of the clique, $v.\eta_{mseen} \geq k$. This is because: (i) v'_1 has been output, (ii) all vertices v'_i in the clique are directly connected to v'_1 with $\eta_{max}(v'_1, v'_i) \geq k$, and (iii) Line 12 of CSV in Figure 4.1 will update $v.\eta_{mseen}$ for all vertices v that directly connects to v'_1 to the maximum of the original $v.\eta_{mseen}$ or $\eta_{max}(v'_1, v)$.

If v is not part of the clique, there must exist some i, $1 < i \leq k$ such that $v.\eta_{mseen} \geq v'_i.\eta_{mseen}$ in order for v to be output before v'_i in the heap. S-ince v'_2,...,v'_k must be in the heap once v'_1 is output, their corresponding value for η_{mseen} must be equal or larger than k. Since $v.\eta_{mseen} \geq v'_i.\eta_{mseen}$, it must be that $v.\eta_{mseen} \geq k$. □

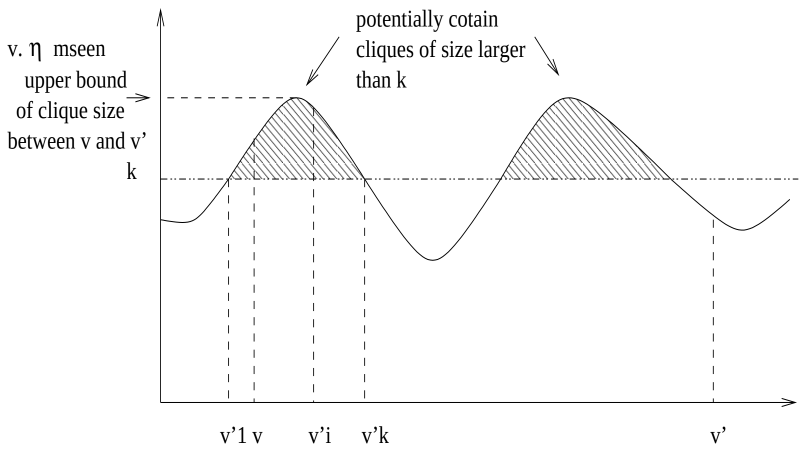


Fig. 4.1: CSV Plot Correctness Proof

Based on Theorem 4.2.1, it is easy to see that any clique with a size larger than or equal to k must fall into a region of the CSV plot in which there exists a continuous set of more than $k-1$ vertices with $\eta_{mseen} \geq k$ in the plot. As an example, Figure 4.1 highlights regions in the CSV plot that potentially contain cliques of size larger than k. To be more precise, the CSV algorithm orders the graph vertices into a CSV plot such that the following theorem can be inferred:

Theorem 4.2.2. *The highest η_{mseen} value between two vertices v and v' on the CSV plot is an upper bound on the maximum clique size that can be found on the subgraph that is induced by considering all the vertices between v and v' in the CSV plot.*

For a clique mining algorithm such as CLAN [WZZ06], the CSV plot provides a good exploratory tool that can highlights regions of interest for mining and also guide parameter settings such as minimum clique size for the algorithm.[1] Later on in our experimental section, we will show that by using the CSV plot as a filtering method, we can speed up the efficiency of CLAN by up to 80% while finding exactly the same set of cohesive subgraphs that the original CLAN algorithm finds.

Finding Cohesive Subgraphs Based on Density

We will next discuss how the CSV plot can be used to provide a solution for Problem 2 which is to find all subgraphs G' such that $|G'| \geq size_{min}$ and $\gamma(G') \geq \gamma_{min}$. We will make use of Turan's theorem [Tur41, Bol78] from extremal graph theory for this purpose.

Theorem 4.2.3. *Turan's Theorem*

[1] The minimum clique size setting in CLAN is used to filter off all cliques that are of a size smaller than a certain threshold

Let $G = (V, E)$ be a graph that contains no clique of size larger or equal to k; then

$$|E| \leq \frac{(k-2)|V|^2}{2(k-1)}$$

Based on Turan's theorem, we can now infer that subgraphs which exceed a certain size and density must contain at least one clique exceeding a certain size. As an example:

Let us assume that $size_{min} = 10$ and $\gamma_{min} = 0.8$. Let G' be a subgraph that just satisfies that criteria i.e., it has 10 vertices and 36 edges (thus ensuring $\gamma(G') = 0.8$). Based on reasoning from Turan's theorem, such a subgraph should contain at least one clique of size 5 (since the 10 vertices graph with max-clique 4 can only have 33 edges maximum). It is also easy to see that larger graphs that satisfy the density threshold must contain cliques of even larger size.

As can be seen, the solution to Problem 1 which helps to identify and visualize cliques of a certain size can be used to provide markers for Problem 2. As any regions in the CSV that do not contain cliques of a sufficiently large size will not contain any subgraphs of interest in Problem 2. Note that this approach of using cliques in a subgraph to measure its cohesiveness is widely accepted in social network analysis [Sei83].

Finding Cohesive Subgraphs Based on Connectivity

We next examine the usefulness of the CSV plot in handling Problem 3. Note that Problem 3 can be converted to a special instance of Problem 2 by observing the following:

1. A k-connected graph must contain at least k vertices.
2. A k-connected graph must contain at least $\lceil (kn/2) \rceil$ edges since each vertex must connect to at least k other vertices.

Based on these observations, we can reduce an instance of Problem 3 into an instance of Problem 2 where $size_{min} = k$ and the minimum density, γ_{min} is set to $\frac{2\lceil (kn/2) \rceil}{k*(k-1)}$. The CSV plot can thus be applied to solve Problem 3 as it is applied to solve Problem 2.

4.2.1 Multi-Dimensional Mapping

As $\zeta_{max}(v)$ and $\eta_{max}(v_i, v_j)$ is expensive to compute due to the clique detection, this section proposes a multi-dimensional mapping which computes an upper bound of these two functions.

Given the graph G, we achieve multi-dimensional mapping by first selecting n vertices as **pivots** based on the shortest path distance. Various methods for selecting these pivots exist [FL95, FTCF01, BNC03]. Here, we adopt a simple strategy of incremental selection. We iterate n rounds and select a vertex that is furthest away (based on average distance) from pivots selected from previous rounds.

Given a pivot p_i, the distance of the shortest path of a vertex v to p_i is denoted as $SD_i(v)$. For an edge e in the graph, we define $SD_i(e)$ as $\frac{1}{2} \times (SD_i(v_1) + SD_i(v_2))$ where v_1 and v_2 are the two vertices connected by e. We can now define mappings of graph elements into n-dimensional space:

Definition 8. M(v), M(e)

Given a graph $G = \{V, E\}$ and a set of pivots p_1,...,p_n, a vertex v will be mapped into the point $M(v) = (SD_1(v), ..., SD_n(v))$ while an edge e will be mapped into the point $M(e) = ((SD_1(v) + SD_1(v'))/2, ..., (SD_n(v) + SD_n(v'))/2)$ where $e = (v, v')$.

Figure 4.2 shows an example of how a graph is mapped into two-dimensional space by picking two pivots v_0 and v_3. The mapping essentially divides the n-dimensional space into grid cells of unit length and vertices are mapped into the intersection of the grid lines with integer coordinates. Edges on the other hand are mapped exactly in between the mapping of the two vertices that they connect. We will use $D_\infty(M(v_1), M(v_2))$ to represent the distance between $M(v_1)$ and $M(v_2)$ under L_∞ norm, i.e.,

Definition 9. $D_\infty(M(v_1), M(v_2))$

$D_\infty(M(v_1), M(v_2)) = max_{i=n}^{i=1}|SD_i(v_1) - SD_i(v_2)|$

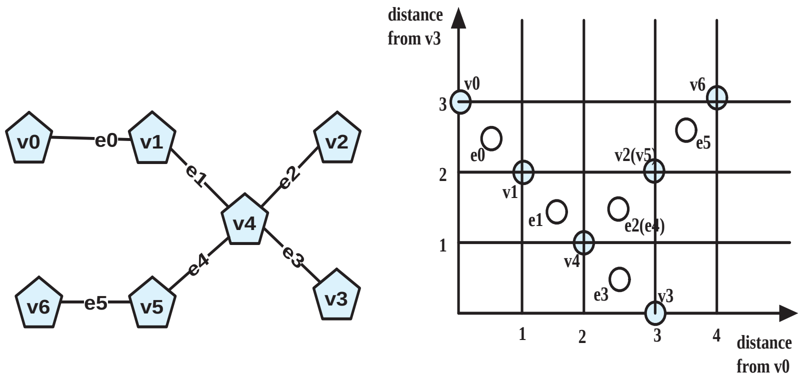


Fig. 4.2: An Example of Graph Mapping

Based on triangular inequality, it is possible to prove the following lemma:

Lemma 4.2.1. *Let the length of the shortest path between two vertices v_1 and v_2 be $D(v_1, v_2)$, then $D_\infty(M(v_1), M(v_2)) \leq D(v_1, v_2)$*

Proof. Assuming otherwise, then there exists a pivot p_i such that $|SD_i(v_1) - SD_i(v_2)| > D(v_1, v_2)$. Without loss of generality, $SD_i(v_1) > D(v_1, v_2) + SD_i(v_2)$, which means that $SD_i(v_1)$ is not the shortest path distance to v_1. This is a contradiction. $\square$

Theorem 4.2.4. *Let $G' = \{V', E'\}$ be a clique of size k in G, then G' will be mapped into a unit grid cell C based on our high dimensional mapping such that:*

- *There exist at least k vertices with a degree greater than $k - 1$ in C.*
- *There exist at least $k(k-1)/2$ edges in C.*

Proof. All vertices in a clique are one edge away from each other. Combining this with Lemma 4.2.1, we know that each pair of vertices v_1', $v_2' \in V'$ will be mapped into n-dimensional space such that $D_\infty(M(v_1), M(v_2)) \leq 1$. Since this must be true for all pairs, the only possibility is that all vertices in the clique are mapped into the same grid cell of unit length. Similarly, since edges are mapped between the two points they connect, they must only be found in the same grid cell. Furthermore, since the vertices belong to a clique of size k, they must at least have degrees of $k-1$ in order to connect to all the other vertices in the clique.

Algorithm 4.2 Algorithm to estimate η_{max} and ζ_{max}

```
Require: Mapping of graph $G$, an empty R-tree $R$
Ensure: $\eta_{max}(v, v')$ and $\zeta_{max}(v)$ Upper bound actual $\eta$ value
 1: set $v.\zeta_{max} = 0$ for all $v \in G$
 2: for each edge $e \in G$ do
 3:     construct set of cells $C$ containing $e$
 4:     for each cell $c \in C$ do
 5:         Add $e$ into $c$
 6:         if $c$ is not found in $R$ then
 7:             Insert $c$ into $R$
 8:         end if
 9:     end for
10: end for
11: for each vertex $v \in G$ do
12:     Locate existing cells set $C$ in R that contains $v$
13:     Add $v$ into every cell $c \in C$
14: end for
15: for each $c$ in R do
16:     Est_$\eta\zeta(c)$
17: end for
18: return $\eta_{max}(v, v')$ and $\zeta_{max}(v)$ $\forall e(v, v') \in G$
```

We will now explain how to compute upper bounds for $\zeta_{max}(v)$ and $\eta_{max}(v, v')$ for each vertex v and each of its connecting vertices v'.

Referring to algorithm 4.2, our algorithm first computes the coordinates of all vertices and edges of the graph after mapping and then insert them into an R-tree[2]. Our R-tree stores unit cells. Each cell has unit length in all dimensions and stores a list of vertices and edges that are mapped into it. To avoid introducing cells that contain only vertices but no edges, we first insert edges into the R-tree and then subsequently the vertices. Complexity arises when an edge is mapped into the boundary of a cell. In that case, more than one cells are added into the R-tree. This happens only when the vertices at the two end of an edge are having the same distance to one or more of the pivots. To minimize such situation, we design the pivot selection method such that the group of pivots are far apart from each other. Experiments on pivot selection methods in later part of this paper support our choice.

After the insertion of all graph elements, the algorithm will compute $\eta_{max}(v)$ and $\eta_{max}(v, v')$ by processing each cell using the algorithm in Figure 4.3.

Given a cell c, algorithm Est_$\eta\zeta$ in Figure 4.3 will compute an upper bounds for $\eta_{max}(v, v')$ for every node v inside c and updates $\zeta_{max}(v)$ or $\eta_{max}(v, v')$ for some v' the upper bound computed are found to be higher than their original value. To achieve this, the algorithm iterates over all vertices mapped inside c. We only start the estimation if c contains enough edges and vertices such that c has potential to update some η_{max} or ζ_{max} values that are relevant to v(i.e. c is

[2] Note that we use an R-tree as it is rather common and readers should feel free to use other spatial indexes for this purpose.

Algorithm 4.3 Algorithm to estimate η in c:Est_$\eta\zeta(c)$

```
Require: A unit cell $c$
1: return Estimate $\eta_{max}$ and $\zeta_{max}$ for vertices in $c$, update value of $\eta_{max}$ or $\zeta_{max}$
   when necessary
2: for every $v$ mapped into $c$ do
3:     if $c$ is sufficiently "dense" to change $\zeta_{max}(v)$ or $\eta_{max}(v, v')$ then
4:         $V' = \{v' | v'$ directly connects to $v$ and $v'$ is mapped into $c\}$
5:         for each $v' \in V'$ do
6:             $V''$ =$\{v'' | v''$ connects to both $v$,$v'$ $\bigwedge v'' \in V'\}$
7:             Let $G''$ be subgraph induced from $V''$
8:             Compute degree array $DV'' \forall v'' \in V''$
9:             such that $DV''(v'')$=$degree(v'', G'') + 1$
10:            $DV'(v')$=$\eta$_bound($DV''(v')$)
11:        end for
12:        $DV = \eta$_bound($DV'$)
13:        for each $v' \in V'$ do
14:            if $\eta_{max}(v, v') < DV'(v')$ then
15:                $\eta_{max}(v, v') = DV'(v'))$
16:                if $\zeta_{max}(v') < \eta_{max}(v, v')$ then
17:                    $\zeta_{max}(v') = \eta_{max}(v, v')$
18:                end if
19:                if $\zeta_{max}(v) < \eta_{max}(v, v')$ then
20:                    $\zeta_{max}(v) = \eta_{max}(v, v')$
21:                end if
22:            end if
23:        end for
24:    end if
25: end for
```

sufficiently "dense" enough for v in line 2). For each neighbor vertex v' of v, we find V'' a set of vertices that are connected to both v and v' and which are also found within c (V'' also contains v, v'). We induce a subgraph G'' that contains all vertices in V'' and all edges in the original graph that join two vertices in V'' (line 5). Based on G'', we compute an array DV'' where $DV''(v'')$ stores the degree of v'' within G''. This is the largest possible clique size v'' can participate in G''. It is

Algorithm 4.4 Function η_bound

```
Require: $DV$: an array of $\eta$ bounds
1: return a new array of tighter $\eta$ bounds
2: $\eta_{new} = 0$
3: S = $DV$ in descending order
4: for $i = 1$ TO $|DV|$ do
5:     if $i \geq S(i)$ then
6:         $\eta_{new} = S(i)$
7:         BREAK FOR LOOP
8:     end if
9: end for
10: for $i = 1$ TO $|DV|$ do
11:     if $DV(v_i) > \eta_{new}$ then
12:         $DV(v_i) = \eta_{new}$
13:     end if
14: end for
15: return $DV$
```

also a loose upper bound of $\eta_{max}(v', v'')$.

Since a clique of size k must contain k vertices each with degree of $k-1$, we will pass DV into a function called η_bound(DV) in Figure 4.4 which will compute an upper bound on the size of the biggest clique that could occur within G''. This is done by sorting DV in descending order, storing it in an array S and going through S starting from the first element until the i^{th} element is greater or equal to $S(i)$. Once the condition is satisfied, i will represent the size of the largest possible clique in G'' and will be use to update η_{new}. Line 9-13 then update DV such that $DV(v_i)$ stores the size of the largest clique that v_i can participate in G''.

Once the function is terminated, Line 7 of Algorithm Est_η will update $DV'(v')$ with the result from function η_bound. $DV'(v')$ thus represent the biggest clique that v and v' can participate in. Our next step is based on the observation that a vertex v that participate in a clique of size k must have at least $k - 1$ direct neighbors that participate in a clique of size k. As such, we pass DV' to function η_bound in Line 9 of Algorithm Est_$\eta\zeta$ to find the largest k, such that are k vertices v' which could participate with a clique of size k together with v. Upon exiting from Est_$\eta\zeta$, DV' could contains a better bound that DV'' on $\eta_{max}(v, v')$ for each $v' \in V'$. The values in DV' are then used to update the value of η_{max} and ζ_{max} for v and v' if the estimated upper bound is higher than the original value. Note that in our algorithm description, we have certain redundancy in that when v' is being processed, the same subgraph G'' will again be induced for edge (v', v). This can be easily avoid by imposing an ordering on the vertices and we omit the description to keep our discussion clean.

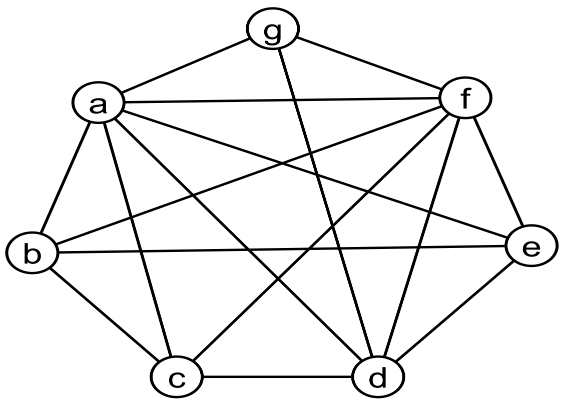


Fig. 4.3: Estimation η between a and Its Neighbors

To illustrate how our algorithm in Algorithm 4.3 works, we present an example in Figure 4.3. To estimate $\eta_{max}(a, f)$, we locate the neighborhood of a and f, $\{a$, b, c, d, e, f, $g\}$. After sorting the degree array in descending order, we have array $\{6(a)$, $6(f)$, $5(d)$, $4(b)$, $4(c)$, $4(e)$, $3(g)\}$ (here we attach the vertex id to each degree value for easy interpretation). Function η_bound(DV) infers from the sorted array that the upper bound of $\eta_{max}(a, f)$ is 5. Similarly, we can estimate an upper bound on η_{max} between a and all its neighbors($\eta_{max}(a, f) = 4$, $\eta_{max}(a, c) = 4$, $\eta_{max}(a, d) = 4$, $\eta_{max}(a, e) = 5$ and $\eta_{max}(a, g) = 4$) and store them in DV'. After calling function η_bound with DV', we tighten the value of $\eta_{max}(a, f)$ to be 4, which is in fact the correct value for $\eta_{max}(a, f)$. Since $\eta_{max}(v, v')$ and $\zeta_{max}(v)$ are independent of CSV traversing order, we can pre-compute them and access them directly when computing the CSV plot. We store the values of all $\eta_{max}(v, v')$ inside a table of $|E|$ entries and all values of $\zeta_{max}(v)$ inside a table of size $|V|$.

Note that since the spatial mapping overestimates the clique size and our approximation computation of η_{max} and ζ_{max} are also upper bound, Theorems 4.1 and 4.2 will apply here. Correspondingly, any CSV plot that is computed based on our approximation method will still be useful for solving Problems 2 and 3.

Complexity Analysis The execution of CSV consists of three parts. The multi-dimensional mapping time, the tree building time and the core algorithm running time.

The multi-dimensional mapping process is the process of computing the shortest path distance between the n pivots and the rest of the vertices. The standard heap implementation has overall time complexity of $O((|V| + |E|) \log |V|)$.

The R-tree that is built on the graph vertices and edges has a complexity of $O((|V| + |E|)log(|V| + |E|))$ if each vertex and edge is inserted once into the tree. The worst case is when each edge is mapped into 2^n grids and so does each vertex. The complexity increase to $O(((|V| + |E|)2^n)log((|V| + |E|)2^n))$. For η estimation algorithm in figure 4.3, if a vertex has degree d_v, the first run of estimation requires d_v rounds of sorting d_v vertices. the total complexity is $O(d_v^2 \log d_v)$. The next run of tightening the upper bound of η requires $O(d_v \log d_v)$. The overall complexity for one vertex is thus $O(d_v^2 \log d_v)$. Due to the uncertainty of mappings, the distributions of the vertices vary. The worst case arises when the graph is a $|V|$-clique. This $|V|$-clique are mapped to 2^d grids identically. Inside each grid, there are $|V|$ vertices and $|E|$ edges. the complexity is thus $O(|V|^2 \log |V| 2^d)$. By checking whether a cell is sufficiently "dense" (as mentioned in algorithm in figure 4.2), the number of cells we actually need to perform η estimation is greatly reduces.

The complexity of the core CSV algorithm depends on the number of comparisons to decide which vertex should enter the stack. Before any vertex is output, all its neighboring vertices are checked. The total complexity is thus $O(|E|)$.

The time complexity for each component of CSV is listed in Table 1. Note that the above scenarios are extreme cases that occurred for extreme graphs. Ex-

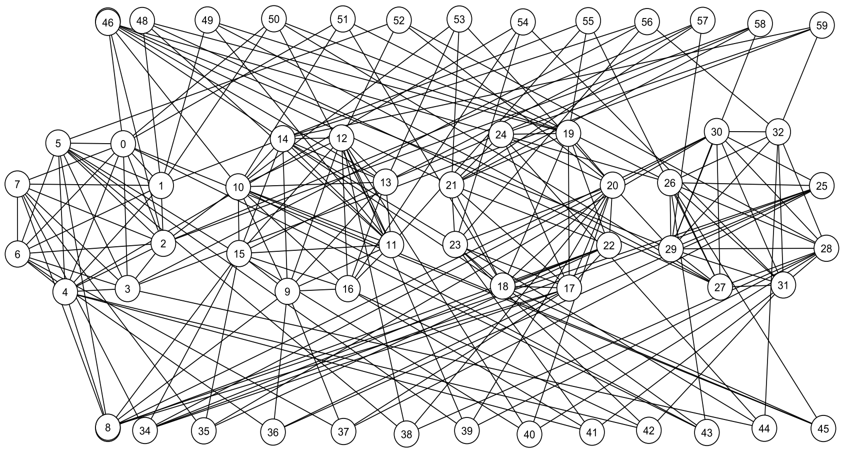
Mapping	$O((\lvert V\rvert + \lvert E\rvert)\log\lvert V\rvert)$
Tree Building	$O(\lvert V\rvert^2 \log\lvert V\rvert 2^d)$
CSV core	$O(\lvert E\rvert)$

Tab. 4.1: CSV Components Complexity

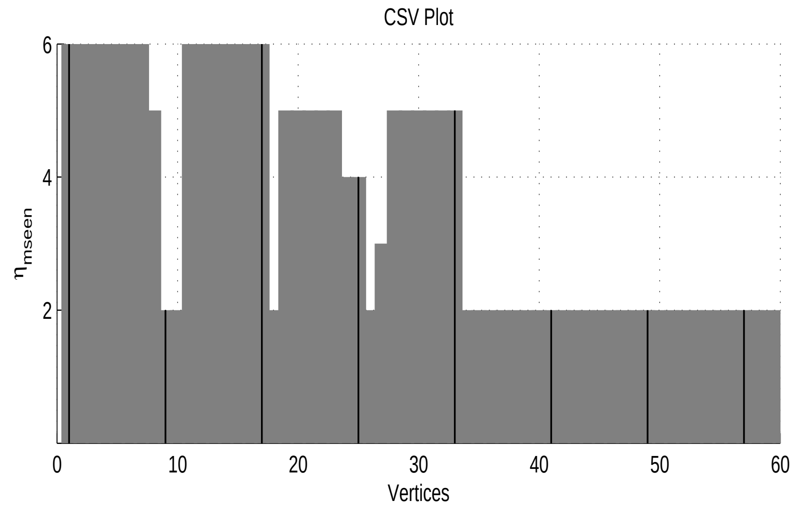
periments on SMD datasets and DBLP datasets shows the number of grid cell are far below the extreme case. Thus our algorithm achieves better performance for real scenarios.

Example Output To give an example on how the output from CSV looks like, we generate a synthetic graph with four cliques of size 8, remove 30% of the edges and then embed the cliques into a random graph. Figure 4.4(a) shows the generated graph with 60 vertices. The numbers shown on the vertices indicate the order of DENSUE's walk on the graph, and the corresponding plot using five pivots for mapping is shown in Figure 4.4(b). Instead of being cliques of size 8, they now become an overlapping set of cliques of size 6 or 7. However, since their connectivity is still higher than the vertices outside the cliques, their presence can still be discerned.

In conclusion, unlike existing "blackbox" algorithms for finding cohesive subgraphs, our visualization plot goes beyond highlighting them to showing their distributions and how they interact with other components in the graph.



(a) Graph With Embedded Partial Cliques



(b) Corresponding CSV Plot

Fig. 4.4: An Example of CSV Plot

4.3 Experiment

Our experiments are evaluated on a Windows-based machine. The machine has P4 3GHz CPU, 1G RAM and 75GB hard disc with Windows XP installed.

We use DBLP 10 years' co-authorship dataset and Stock Market(SMD) data used in algorithm evaluation in [ZWZK06] to evaluate the effectiveness and efficiency of CSV. The DBLP data set covers co-authorships across year 1997 to 2006. Each graph vertex represents an author and two authors are connected by an edge if they co-authored in at least one publication within a year. We consider the co-authorship relations is significant only when the two researchers work together for at least two years. The compound DBLP 10-year co-authorship graph contains 2819 vertices and 54990 edges.

The SMD data is a collection of three sets of 11 graphs from [ZWZK06] which are named SMD-95, SMD-93 and SMD-90 respectively. Here, 0.95, 0.93 and 0.90 are correlation thresholds and an edge exists between two stocks in the graphs if their correlation is found to be above the correlation threshold. Table 4.3 shows the statistics of these three sets of graphs. For each set of 11 graphs, we create a summary graph in which an edge exists between two stocks if it is found in more than a certain number of graphs. Depending on the threshold applied to each of the datasets, the final summary graphs' size varies. For all our evaluation, the number of pivots are set to 4 unless otherwise stated. We abstract the largest connected component from each data set and performs CSV and CLAN on it.

Datasets	# Graphs	Avg. # of vertices	Avg. # edges
Stock Market 0.95	11	1683	20074
Stock Market 0.93	11	2618	68608
Stock Market 0.90	11	3636	206747

Tab. 4.2: Stock Market Datasets (SMD) Statistics

Sup.	SMD-95		SMD-93		SMD-90	
	V	E	V	E	V	E
1	4264	132359	5323	403190	6008	1064133
2	3409	55945	4617	200882	5498	613823
3	2373	16462	3809	88301	4931	314421
4	1600	6893	2968	35892	4263	150692
5	425	1680	2148	14368	3587	69412
6	237	756	669	1771	2760	31462
7	84	315	361	723	1024	5611
8	65	183	219	364	635	3032
9	45	99	93	504	450	1681
10	13	20	214	409	356	1009
11	10	12	123	230	242	522

Tab. 4.3: Statistics of Largest Connected Components for Stock Market Datasets' Summary Graphs(SMD)

4.3.1 Effectiveness of CSV Plot

In this section, we will look at the effectiveness of the CSV plot.[3]

4.3.1.1 DBLP Plot

We first test CSV on the DBLP dataset. As an instance of social networks, DBLP data set reflects various social processes such as information processing, distributed search and diffusion of social influence [KW06]. We experiment on the largest connected component of the compound ten years' DBLP co-authorship data and show it's CSV plot in Figure 4.5.

[3] In support of the SIGMOD'2008 experimental repeatability requirement, the ordering of the DBLP (Figure 4.5) and stock(Figure 4.10) datasets together with their η_{mseen} values are made available at:
http://www.comp.nus.edu.sg/˜atung/publication/cri.zip

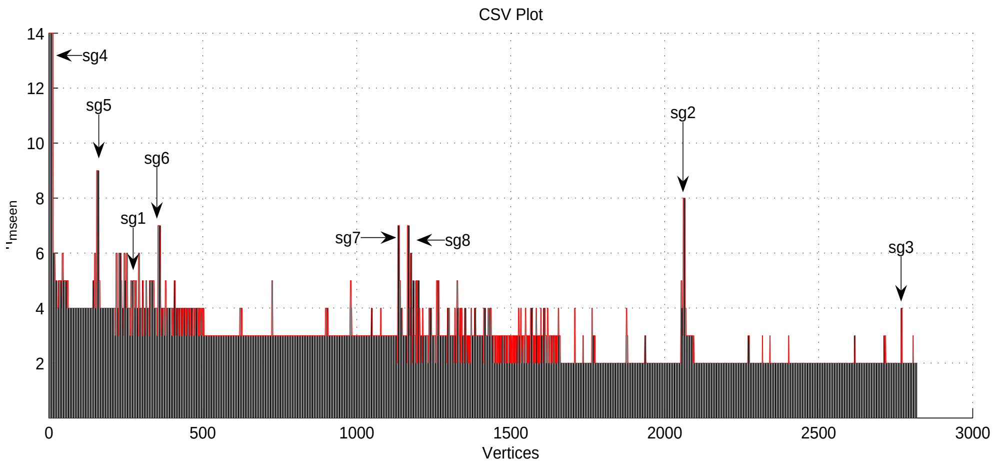


Fig. 4.5: CSV Plot for DBLP 97-06 Co-authorship Graphs

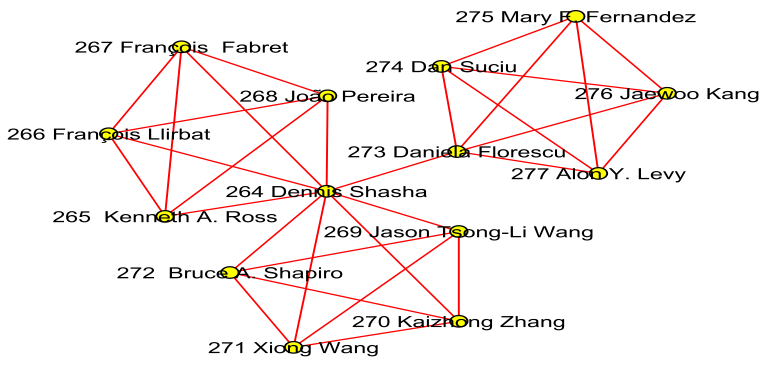


Fig. 4.6: Cliques Joined by 2 Co-authors in $sg1$

As an example to show how the distribution of cliques are depicted by the CSV plot, we show subgraph $sg1$ in Figure 4.6 which consist of vertices ordered from 264 to 277 in the CSV plot. As can be seen from the plot, there are multiple small peaks rising from an otherwise flat region. These are in fact small cliques that are joined together by some co-authors from each clique. In this case, $sg1$ contains three cliques that are joined together by 2 authors, namely, Dennis Shasha and

Daniela Florescu. Here, Dennis Shasha is a member of two of the cliques while Daniela Florescu is a member of the remaining clique with all three cliques having a size of 5.

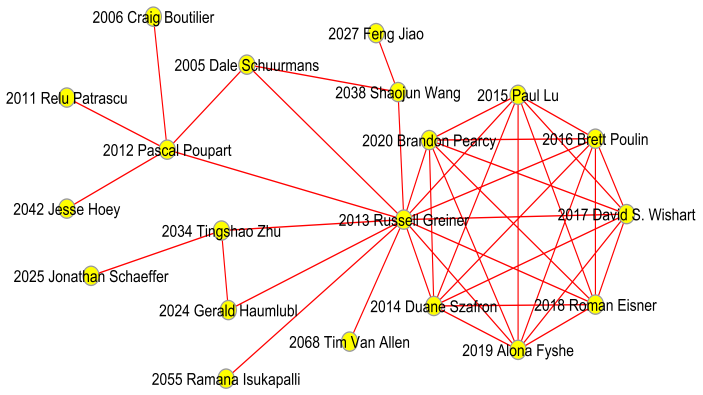


Fig. 4.7: A large Clique in $sg2$

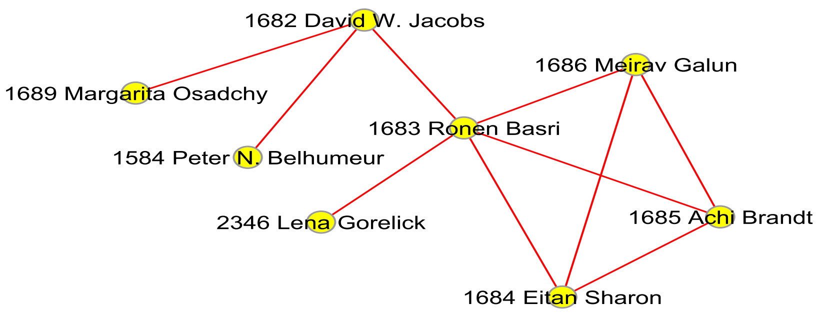


Fig. 4.8: A Small Clique in $sg3$

Next, we will show an example of a highly connected subgraph marked as $sg2$ in Figure 4.5 and displayed in Figure 4.7. Since the peak of $sg2$ seems to featured prominently in a neighborhood with low cohesiveness, we also abstract some partial neighbors of $sg2$ to confirm our suspicion. As can be seen from Figure 4.7, $sg2$ in fact represent a group whose member are Russell Greiner, D-uane Szafron, Brett Poulin , David S. Wishart, Roman Eisner, Alona Fyshe and Brandon Pearcy. Within the CSV plot, they are ordered consecutively from 2013 to 2020. From the snippet in Figure 4.8, we can identify a “hub” person: Russell Greiner who is ordered as the first person within $sg2$. Indeed he is a well-known researchers in areas such as Bayesian Networks and leads several research groups which explains his important role in linking up part of the DBLP co-authorship graphs. In general, a sudden raise in the CSV plot usually indicates a key vertex whose removal may greatly affects the density distribution of the graph. Note the “sparseness” in other part of the graph in Figure 4.7 which corresponds to the low density region around $sg2$ in the CSV plot.

As we have stressed, the strength of the CSV plot is not only on its ability to find cliques but to also present the overall density distribution of a graph. As can be seen from Figure 4.5, cliques of size 4 are in fact available in abundance in the DBLP graph and a “blackbox” pattern mining algorithm will have found many cliques of size 4 without knowing their relationship with their neighborhood. From the CSV plot however, we can see that the subgraph $sg3$ is prominent within its neighborhood despite it being just one of the many size 4 cliques in the graph. The subgraph $sg3$ (Figure 4.8) consists of vertices ordered from 1683 to 1686 in Figure 4.5 and researchers in the group include: Ronen Basri , Eitan

No.	Size	Order	Members
$sg4$	14	0-13	J. Miller, S. Sudarshan, M. Nemeth, Rajeev Rastogi, Jerry Baulier, Abraham Silberschatz, Peter McIlroy, P. P. S. Narayan, Henry F. Korth, A. Khivesera, C. Gupta, Philip Bohannon, S. Joshi, S. Gogate
$sg5$	9	153-161	Volker Markl, Rudolf Bayer, Timos K. Sellis, Roland Pieringer, Klaus Elhardt, Frank Ramsak, Robert Fenk, Aris Tsois, Nikos Karayannidis
$sg6$	7	352-358	Mitch Cherniack, Michael Stonebraker, Ugur C-cediletintemel, Anurag Maskey, Stanley B. Zdonik, Nesime Tatbul, Donald Carney
$sg7$	7	1131-1137	Bernhard Schoumllkopf, Thomas Navin Lal, Wolfgang Rosenstiel, Michael Schroumlder, N. Jeremy Hill, Thilo Hinterberger, Niels Birbaumer
$sg8$	7	1162-1168	Peter M. G. Apers, Martin L. Kersten, Henk Ernst Blok, Roelof van Zwol, Willem Jonker, Milan Petkovic, Menzo Windhouwer

Tab. 4.4: Large Cliques in DBLP

Sharon, Achi Brandt and Meirav Galun. These four researchers' major research interests are in computer vision and maths. Besides common research interests, they all currently work in Israel. Without prior background information of these researchers, we will not identify this group of researchers from other cliques of size 4 in the DBLP graph.

Finally, we show the remaining cliques that are of size 7 and above in Table 4. Among these cliques, $sg7$ and $sg8$ are the closest to each other on the CSV plot. This can be explain by the fact that both groups essentially work in Germany. However, $gp7$ consists of members who do research in AI while $gp8$ consists of members who do research in databases.

To assess the accuracy of our estimation for η_{max} and ζ_{max}, we also plot in

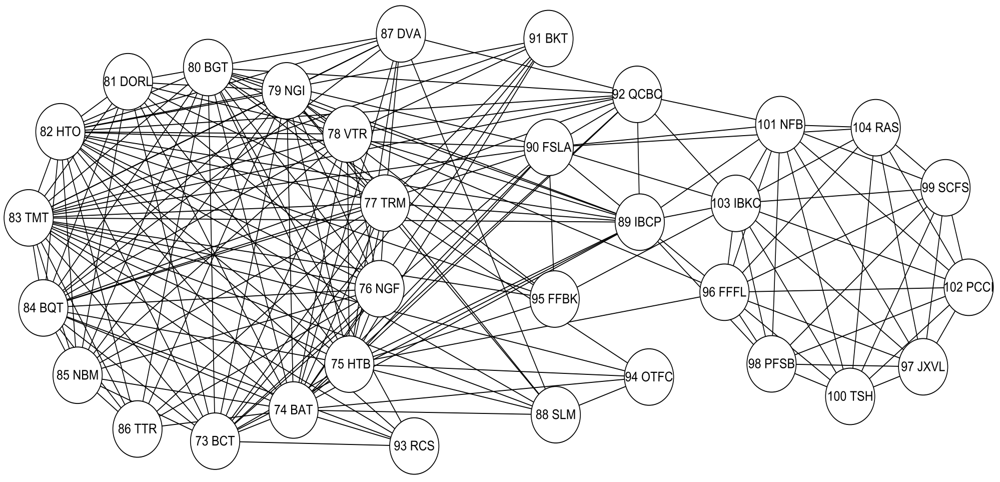


Fig. 4.9: Two Groups of Highly Cohesive Stocks

red the actual η_{mseen} computed by CLAN in Figure 4.5. As can be seen, our estimation of the η_{max} and ζ_{max} are highly accurate resulting in plot that show us the various interesting discoveries that we have discussed earlier.

4.3.1.2 Stock Market Data

We also run CSV on SMD-95 with a support threshold of 45% (i.e. 5 graphs) and generate a CSV plot shown in Figure 4.10. From the CSV plot, we identified two closely related cliques which are at position 73 to 104.

The subgraph induced from the stocks within this region are shown in Figure 4.9. From the figure, we can see two distinct cohesive sub-components that are yet closely related. Such form of nested structures will not be easily detected by normal "blackbox" pattern mining algorithm but is easily indicated in the CSV

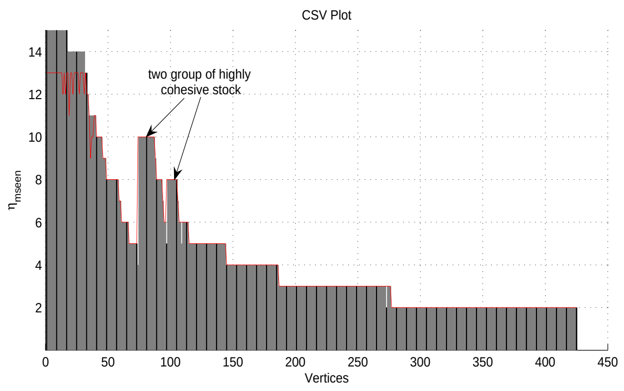


Fig. 4.10: 4D SMD95 CSV Plot with 45% Support Threshold

plot. The larger components consists of 14 stocks participating in 10-cliques. Out of the 14 stocks, 5 stocks belong to BlackRock Group: BlackRock Broad(BCT), BlackRock Advantage Term Trust, Inc(BAT), BlackRock Global Investment Management (BGT), BlackRock Investment Quality Term Trust(BQT), BlackRock Municipal Target Term Trust (TTR). Anther well-known investment bank Merrill Lynch (NBM) announced in Feb 2006 that it would combine with BlackRock to form the World's Largest Independent Investment Management Firms. The rest of the 8 stocks are in areas of investment management as well. Two stocks in fact are the same trust fund over different years(NGI and NGF are 2003 and 2004's national Government Income Term Trust respectively). We thus infer that this large dense subgraph is formed mainly by BlackRock Group and Merrill Lynch with other stocks in investment management field. The other smaller component (9 stocks forming 8-cliques) has four bank and financial service stocks: Fidelity Bancorp (FFFL), Seacoast Financial (SCFS), Jacksonville Bancorp (JXVL) and

PennFed Financial (PFSB). Both the components have interest in insurance and as such they are linked by some health service stocks such as Davita (DVA). CSV plot makes it possible to identify how closely related these stocks are to each other without drilling down to individual stocks.

Like in the DBLP case, we also indicated in red the actual η_{max} values in Figure 4.10. As can be seen, our estimation of the clique sizes are rather accurate. We also investigate how the CSV plot of SMD-95 is affected when varying the number of pivots by computing the CSV plot with 6, 8, 10 and 12 pivots. However, due to the effectiveness of our bounding method, these plots look mostly the same. As such we show only the plot for 12 pivots here in Figure 4.11.

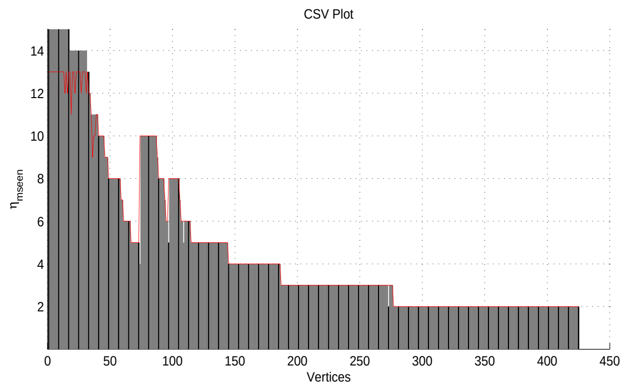


Fig. 4.11: CSV Plot with 12 Pivots

4.3.2 *Efficiency*

4.3.2.1 *Graph Size and Running Time*

By applying CSV on a set of relevant graphs with different sizes, we are able to see how the sizes of the graphs affect the running time of CSV. Experiments are ran on three sets of stock market data (SMD-90, SMD-93 and SMD-95) with different support thresholds to vary the size of the graphs. Readers are referred to Table 3 for the size of the largest connected component summary graphs. Note that the number of edges doubles for every decrease of 1 in the support threshold.

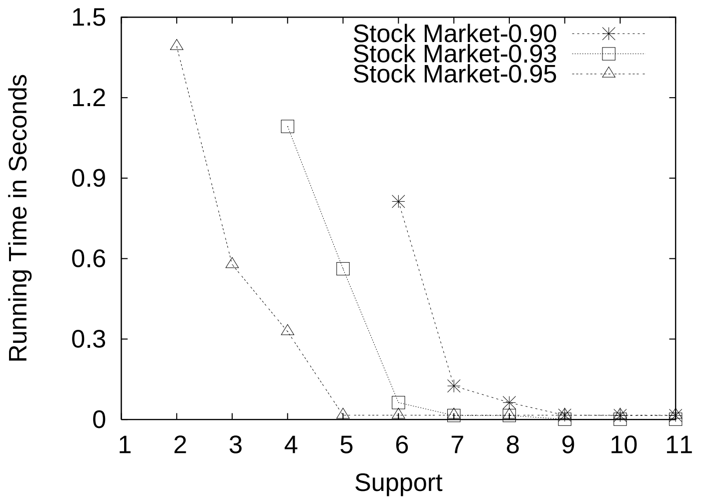


Fig. 4.12: 4D Mapping Time

Figure 4.12, 4.13 and 4.14 indicate the running time of the three components of CSV on the three sets of SMD data. In Figure 4.12, we show the spatial mapping time for the three sets of data. As stated previously, the mapping is performed

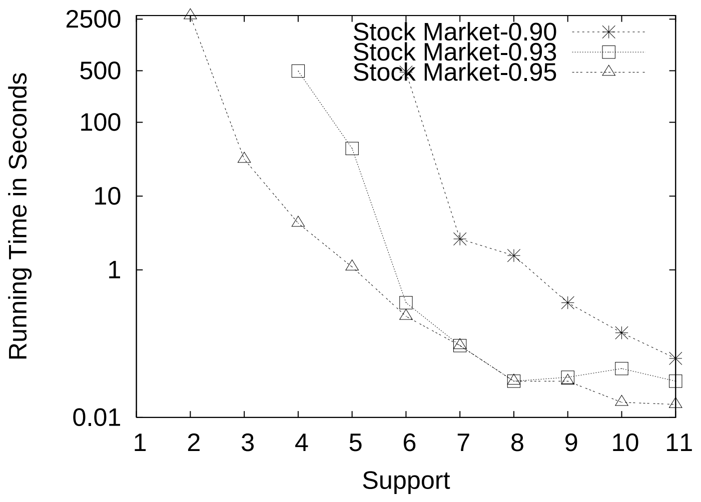


Fig. 4.13: Tree Building Time

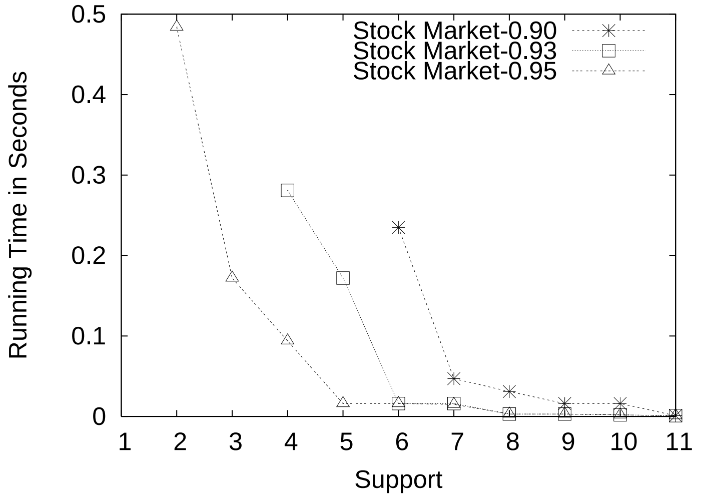


Fig. 4.14: CSV Core Algorithm Running Time

on the largest connected components of the graph. The mapping time ranges from 0.015 sec to 1.391 seconds for graph with size 10 vertices and 13 edges to graph with 3409 vertices and 55945 edges. The mapping process only takes up 0.25-1% out of total CSV running time for a large scale graph. The major time consuming part of CSV algorithm is the tree building process. Figure 4.13 shows the time spent on building the R-trees on the same datasets. The building time does not exceed 2500 seconds for the largest graph that we have process (smd95sp2 with 3409 vertices and 55945 edges). CSV core algorithm's running time on SMD datasets is shown in Figure 4.14. After building up the tree structure, the exploration process is within a second. Note that sometimes the running time of CSV is not monotonically increasing with the graph sizes. The distribution of the edges inside a graph also determines the running time. Graphs with similar number of vertices and edges may be mapping into different number of grids. It is the number of grids and number of elements inside the grids that determine the total running time of CSV algorithm.

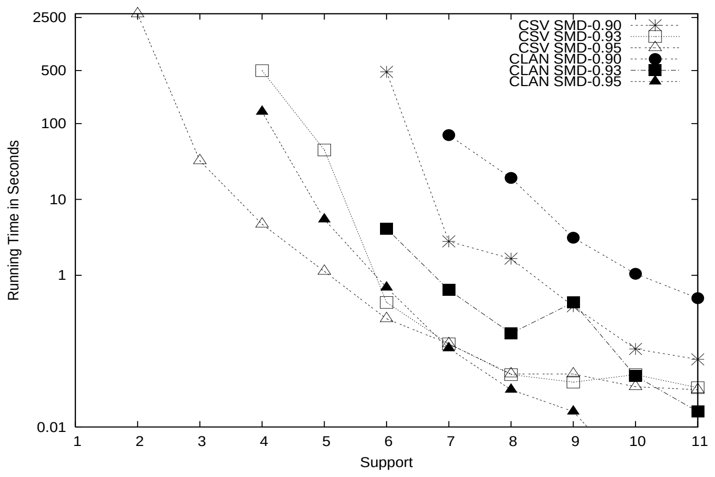


Fig. 4.15: CSV vs. CLAN Running Time

We also compared the overall running time of CSV (Mapping + Tree Building + Core Algorithm) with that of CLAN on SMD data sets. From Figure 4.15, CSV outperforms CLAN both in terms of capability and efficiency. CSV is able to handle graphs 2-4 times larger than CLAN while the running time is only 10%-1% of CLAN when dealing with moderate to large graphs. CSV proves itself to be a good density estimation tool compromising little accuracy with an order of magnitude time savings over exhaustive mining algorithm like CLAN.

4.3.2.2 Pivots Selection Algorithm and Their Effect on Running Time

Although the number of pivots does not affect the accuracy of the CSV plot significantly, we are still interested in the cost when varying the number of pivots. Thus we investigate the effect of the number of pivots selected on the running time over the same SMD-90 datasets. Figure 4.16 presents the total running time comparison of CSV algorithm on 4D to 12D multi-dimensional points of SMD-90 datasets (with support from 8 to 11). As we increase the number of pivots, the mapping dimension increases accordingly, which results in exponential increase in the number of non-empty grids. During the process of mapping, we are only interested in grids with edges in it. A large portion of grids thus become empty and CSV does not need to spend time exploring those grids. The running time is thus not exponentially increasing with number of pivots. This suggests that future users should be cautious when choose more pivots in the mapping process. Our suggestion here is not to use more than 6 pivots.

The algorithm for selecting the pivots also affects the efficiency of CSV. Fig-

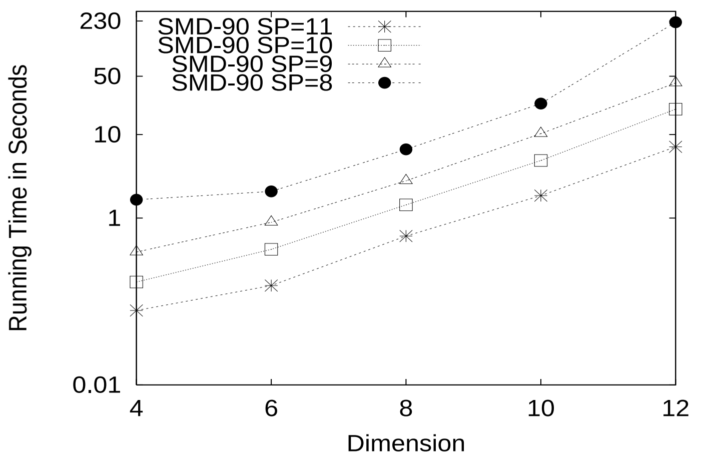


Fig. 4.16: CSV Core Algorithm Running Time Varying Dimensions

ure 4.17 shows three lines representing three different approaches for selecting 4 pivots on the SMD-95 dataset. The "Random" approach simply picks the pivots randomly. The "Separated" approach is the one we discussed in previous sections while the "Central" approach is to pick the pivots with the minimum distance to its furthest vertex. Instead of directly plotting the algorithm's running time for the three different approaches, we plot the number of non-empty grid cells that are being handled in the algorithm. This quantity better reflects the pivots' quality. Results are averaged over 5 runs to off-set the randomness of the first approach. From Figure 4.17, the number of visited grids is significantly reduced by carefully picking the pivots.

In conclusion, our experiments show that CSV is an efficient and effective tools for visual mining and exploration of dense subgraphs. CSV provides a very useful alternative to the "blackbox" and exhaustive enumeration approach of other graph pattern mining algorithms [ZWZK06, WZZ06, HYH+05, YZH05], which often have more parameters to specified as well.

4.3.3 CSV as a Pre-selection Method

Since CSV provides a method to quickly estimate a graph's density distribution and CLAN is an exact algorithm computing graph's max closed cliques, we combine the two algorithms in the hope to achieve fast and accurate computation of graph density distribution. In this section, we first apply CSV on the stock market data to obtained an estimation of graph vertices density. After that, we select those vertices that could potentially contain cliques of required size and apply CLAN on the subgraphs that is induced by these vertices. As expected, CSV does not miss any cliques based on what we have proved earlier. Figure4.18 shows the running time of such an approach compared with that of directly applying CLAN

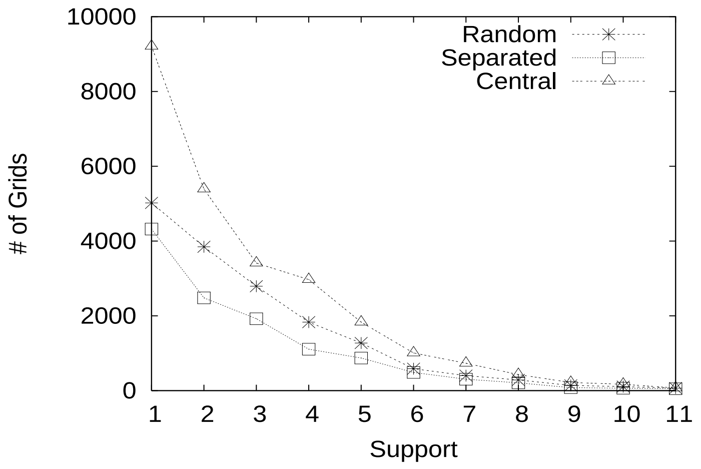


Fig. 4.17: Three Different Pivot Selection Schemes and Resulting #Grid

on the data set. This set of experiments are run on 11 sets of stock market data. The x-label of the graph indicates the characteristics of the data set. For example, ”90-11-5” denotes smd90 data with absolute support 11 and the minimum clique setting is set to 5 (i.e. we want to find all cliques of size 5 and above). The results show that running CSV as pre-selection method for CLAN saves 23% to 84% of the time compared to running CLAN alone.

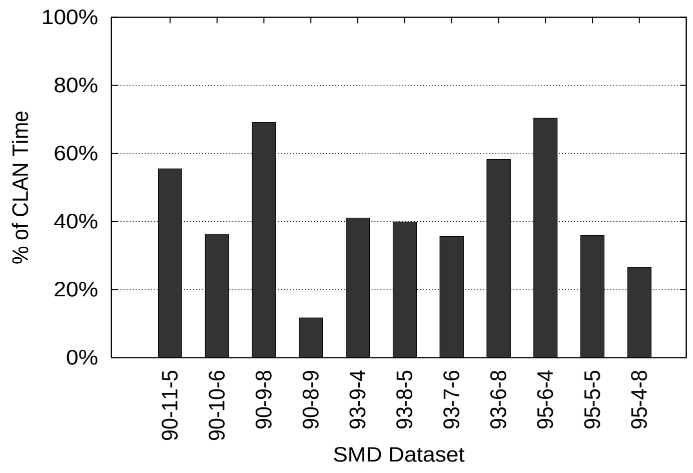


Fig. 4.18: Efficiency of CSV as a Pre-selection Method

Note that the same method here can be used to find the top-k largest cliques in the graphs by iteratively selecting the highest peak on the CSV plot and running CLAN on the region around the peak. We plan to explore more towards this direction in the future.

4.4 Chapter Summary

In this chapter, we propose CSV , an algorithm for mining and visualizing cohesive subgraphs. Existing approaches to the problem typically perform an exhaustive enumeration and output a set of cohesive subgraphs which are often difficult to correlate and understand. CSV relies on a locally measurable notion of density coupled with novel mapping function to visualize and mine cohesive subgraphs. We demonstrate the efficacy and efficiency of CSV on two real datasets. As an algorithm that executes in polynomial time, CSV can be useful as a tool for general exploration of a graph before a region of interest can be selected for more detailed analysis. Additionally we demonstrate that one can use the CSV plot as a pre-filtering method, to speed up the efficiency of clique mining algorithms such as CLAN by up to 80% while finding exactly the same set of cohesive subgraphs as the original algorithm (CLAN) does.

5 On Triangulation-based Dense Neighborhood Graphs Discovery

As presented in section 4, mining for dense patterns are primary tasks in many applications. When the graph size keeps growing, the dense patterns become harder to be located. To address this difficulty, This paper introduces a new definition of dense subgraph pattern, the DN-graph. DN-graph considers both the size of the sub-structure and the minimum level of interactions between any pair of the vertices.

The mining of DN-graphs inherits the difficulty of finding clique, the fully-connected subgraphs. We thus opt for approximately locating the DN-graphs using the state-of-the-art graph triangulation methods. Our solution consists of a family of algorithms, each of which targets a different problem setting. These algorithms are iterative, and utilize repeated scans through the triangles in the graph to approximately locate the DN-graphs. Each scan on the graph triangles improves the results. Since the triangles are not physically materialized, the algorithms have small memory footprint.

With our solution, the users can adopt a “pay as you go” approach. They have the flexibility to terminate the mining process once they are satisfied with the quality of the results. As a result, our algorithms can cope with semi-streaming environment where the graph edges cannot fit into main memory. Results of extensive performance study confirmed our claims.

The rest of the chapter is organized as follows: First we explain the motivation for the triangle based DN-graph mining in section 5.1. In section 5.2, we review recent research effort. The definition of DN-graph is formally presented in section 5.3. This section also highlights DN-graph's features with illustrative examples. Algorithms TriDN and BiTriDN are presented in section 5.4. In section 5.5, the paper describes the semi-streaming graph model. Following that, a semi-streaming solution is presented. Experimental studies are then described in Section 5.7. Section 5.8 concludes this chapter.

5.1 DN-graph Mining, the Motivation

Graphs are the most pervasive model of entity interactions as it concisely captures the interactions among entities. However, for large graphs (which are becoming increasingly common in many applications), it becomes too complicated for human beings to find key information without the help of suitable graph mining technology. Graph mining refers to the process of discovering designated subgraphs from a target graph, in the hope of uncovering unknown knowledge about the graph. When facing unsolvable resource constraint, how to answer the mining question to the best, becomes more challenging.

Most recent works on graph mining [ABC$^+$04, ARS02, ATH03, BBP06, HYH$^+$05] believe that dense patterns are prominent. They capture the most active involvement of entity interactions. Subsequently, researchers propose various definitions of dense substructures. In section 3.2, we review some of these patterns.

Intuitively, a dense pattern contains a set of highly relevant vertices. They usually share large number of common neighbors (two vertices are neighbors if they connect to each other by an edge). The definition of DN-graph (a.k.a. **Dense Neighborhood graph**) follows this intuition.

This paper provides a set of algorithms to mine DN-graphs from large scaled graphs. The problem of mining DN-graphs is an NP-complete problem (due to the close relationship between DN-graphs and cliques, lemma 5.3.1 will cover this in detail). As such, in this paper, we opt to design approximate solutions. In our solutions, the local neighborhood size is the most important, but difficult, quantity to be computed. We associate this quantity with local triangle counting in order to approximate it efficiently.

Graph triangulation refers to the process of generating all triangles in a graph. Our approach locates DN-graph by using the state-of-the-art triangulation algorithm [SW05, BBCG08]. As the storage of triangles can be expensive, we do not store these triangles. Instead, we design our approach to operate iteratively. In each iteration, our scheme dynamically regenerates all triangles and improve the connectivity estimation between vertices in each round.

Such an iterative, triangulation-based approach has three advantages. Firstly, most of the details involved in efficient processing, such as minimizing I/Os, are abstracted within the triangulation algorithm. The abstraction ensures our approach's extensibility to different input settings, e.g. when the target graph is too large to fit into memory, our approach only needs to change the access method of the graph links. In addition, the estimation of the local neighborhood is encapsulated within the triangulation algorithm. Secondly, as the estimation of the local density value improves with each additional iteration, users can adopt a "pay as you go" approach and obtain the most updated results on demand. Finally, when the graph is too large to fit into the main memory, we can collect statistics in the first iteration to support effective buffer management, should there be a need to store the local density value on a disk, since the triangles are generated in the same ordering in every iteration.

	In Memory	Time	Space
TriDN	Yes	$O(klog\|V\|\|E\|^{\frac{3}{2}})$	$O(\|V\|log\|V\| +\|E\|)$
BiTriDN (Binary Bounding)	Yes	$O(klog\|V\|\|E\|^{\frac{3}{2}})$	$O(\|V\|log\|V\| +\|E\|)$
StreamDN (Semi-Stream)	No	$O(k\|E\|)$	$O(\|V\|)$

Tab. 5.1: A Family of DN-Graph Mining Algorithms

In this chapter, we present triangulation based dense graph mining algorithms. Together they form an algorithm family. Their key features are compared in Table 5.1. For brevity, we name them respectively as 1) TriDN, 2) BiTriDN and 3) StreamDN.

Algorithms TriDN and BiTriDN are two variances that handle in-memory graphs. Both algorithms iteratively generate triangles to refine the λ value. These two processes reach convergence when all λ values remain the same as previous iteration.

The third algorithm, StreamDN, is for semi-streaming graph setting. In section 5.5, we introduce the model of semi-streaming graph. To mine semi-streaming graphs, algorithm StreamDN applies the min-wise independent set property, which provides an approximation for triangulation using sequentially scan of graph edges, with bounded error.

5.2 Dense Patterns Mining and Triangulation

A dense graph pattern is a connected subgraph that has significant internal connections with respect to the surrounding vertices. (Depending on the semantic meaning of the graph data, various forms of dense patterns have been investigated in the literature. Please refer to chapter 3 for various types of dense patterns). In this chapter, a **dense subgraph** is a set of vertices sharing many common neighbors. If two connected vertices share one common neighbor, they form a triangle together with their common neighbor. In view of the association between dense patterns and triangles, we further study the problem of triangle counting.

Triangle counting and listing have been well studied in the literature. Given a graph G with $|V|$ vertices and $|E|$ edges, [SW05] proposed a triangle-listing algorithm with time complexity $O(|E|^{\frac{3}{2}})$ and with O($3|E| + 3|V|$) space. Further work [Lat07] improves the performance of the algorithm by separating the vertices into two types, dense and sparse. The improved technique has the same time complexity as the work in [SW05] while it reduces the space complexity to $O(|E| + |V|)$. The above ideas count triangles by scanning graph edges, and join adjacency list of the two vertices. The scanning of graphs makes these techniques highly adaptable to streaming environment (In section 5.5, we discuss the graph streaming model.).

Other research works on mining dense subgraphs can be classified according to their counterparts in item-set mining approaches. The most relevant work is the density based solution [WSTT08]. This work provides not only a way to find the closed cliques (biggest clique among the neighborhood) but to order all graph vertices into a linear fashion for visualization purpose. One of the leading approaches in [GRT05] adopts two-level-shingling method. Although the work only demonstrates its power in collecting statistics from extremely large graph, its performance is impressive and this approach can be employed into graph mining domain to handle large scale graphs.

5.3 DN-Graph as a Density Indicator

This section presents how we characterize interesting patterns inside graphs. Before that, we introduce the symbols used in this paper.

A graph $G(V, E)$ consists of a set of vertices V and a set of interactions E over $V \times V$. The size of G, denoted as $|V|$, is the number of vertices in V. The neighborhood of a graph vertex v, is the set of vertices directly connecting to v. We use $N(v)$ to represent it. If vertex u and v share some common neighbors, we use $N_{\cap}(u, v)$ to represent the joint neighborhood. The neighborhood of e is the joint neighborhood of its two end vertices. We denote the joint neighborhood as N_e. For a subgraph G' of G, the neighborhood of G', $N(G')$, is the set of vertices $u \in G \setminus G'$, which immediately connect with vertices in G'. Inside a graph, the measurement of minimal joint neighborhood size between any connected vertex pair is denoted as λ. We use the notation $\lambda(G)$/$\lambda(V)$ to refer to the measurement of a graph G with vertex set V. For brevity, we omit the content inside bracket and use λ when the context is clear. We also use $\tilde{a}$ to represent an upperbound of quantity a. The upperbound of λ is thus written as $\tilde{\lambda}$.

As mentioned in previous chapters,a clique is a fully connected graph, in which every pair of vertices are connected by an edge. If the size of a clique is c, we call the clique a c-clique. When compared with clique of the same size, a quasi-clique has only a fraction (say δ) of edges in the graph, it is a δ quasi-clique.Conventionally δ is in the interval $(0.5, 1]$.

Definition 10. *DN-Graph*

A DN-graph with parameter λ, denoted $G'(V', E', \lambda)$, is a connected subgraph $G'(V', E')$ of graph $G(V, E)$ that satisfies the following conditions: (1) Every connected pair of vertices in G' share at least λ common neighbors.
(2) for any $v \in V \setminus V'$, $\lambda(V' \cup \{v\}) < \lambda$; and for any $v \in V'$, $\lambda(V' - \{v\}) \leq \lambda$.

As the definition states, a DN-graph should be a connected subgraph in which the lower bound of shared neighborhood between any connected vertices, λ, is locally maximized. Being a DN-graph, it has local maximal λ value and the size of the DN-graph is maximized. This ensures that the DN-graph has more distinguishing power and maximal coverage. Similar with the graph's diameter

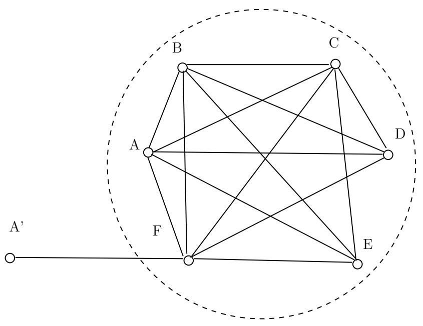
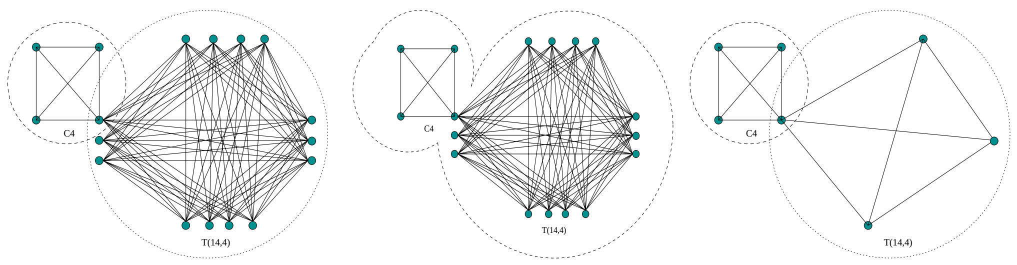


Fig. 5.1: A DN-graph

and minimum cut, λ is an indicator of the graphs' underlying density.

As proven in Theorem 5.3.1, it is a local maximum graph. For example, in figure 5.1, subgraph $ABCDEF$ is a DN-graph of λ value 3. If we include one more vertex A', the λ value of the graph $A'ABCDEF$'s drops significantly to 0. Similarly, taking away any vertex, say A, leads to a lower value λ. For further illustration, we compare DN-graph with different dense patterns in next subsection.

5.3.1 *An Illustrative Example to Compare Different Dense Patterns*



(a) C4,T(14,4) (b) δ Quasi clique (c) Max Clique

Fig. 5.2: A Graph and Its Different Dense Sub Structures

Figures 5.2(a) to 5.2(c) present mining results of two different density criteria on an illustrative graph. The graph in Figure 5.2(a) contains a 4-clique loosely attached to a Turan's graph. A Turan's graph $T(M, N)$ is a special class of graph, in which N graph vertices are divided equally (or as equal as possible) into N groups. Every pair of vertices from two different groups has edge connecting them, while members in the same group are not connected. In fact, Figure 5.2(a) embeds a Turan's graph $T(14, 4)$. Judging by the interactions, there are two interesting substructures in Figure 5.2(a). One is the 4-clique and the other is the Turan's graph. If we apply δ quasi-clique mining, when setting $\delta = 0.8$, the mining result includes the whole example graph (figure 5.2(b)). The mining results are so relaxed that two dense substructures cannot be distinguished. If we insist on regularity of the pattern (e.g. a clique), we can only find subgraphs of size 4 (as indicated in 5.2(c)) while missing the Turan's graph.

DN-graph is designed to represent dense patterns, as it captures subgraphs with more internal associations. It is thus not surprising to see the correlation between DN-graph's λ value and maximum clique size, which is another popular dense indicator. In the following subsection, we demonstrate above facts using a dynamical graph with increasing edge size.

5.3.2 λ Value and Clique Size Changes inside a Dynamic Graph

Figure 5.3 illustrates the relationship between λ value and clique size inside a dynamic graph. The graph consists of 20 separated vertices and initially has no

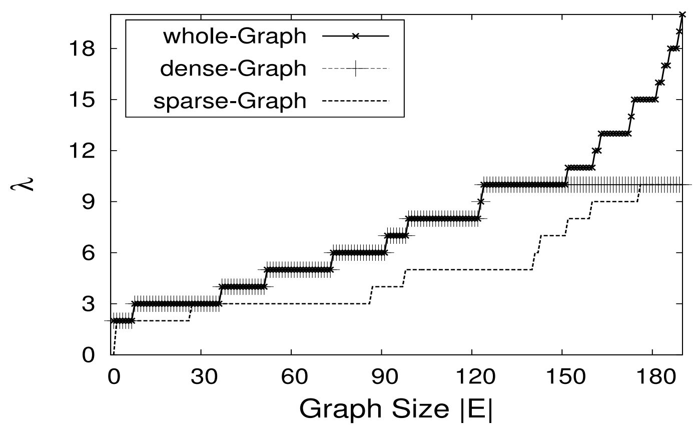


Fig. 5.3: The Growth in λ of a 20-Vertex Dynamic Graph

edges. The dynamic graph varies its topology by adding one edge each time. For comparison purpose, the vertices are deliberately separated into two groups. Each group consists of 10 vertices. One group ("dense") has much higher probability of adding a new edge. The other group's probability of new edge appearing is significantly lower, thus is considered as "sparse" sub graph. As the edge keeps on adding, the whole graph becomes a 20-clique. Not surprisingly, the "dense" group becomes a 10-clique (when adding up to 123 edges) faster compared with "sparse" sub graph. Correspondingly, the λ value of the "dense" group grows substantially faster than the "sparse" one. From this example, the λ value indicates dense sub pattern well. With the illustrative example, we next analyze the relationship between DN-graph and a well-known dense pattern: the closed clique.

5.3.3 Relationship between DN-graph and Closed Clique

Similar with the graph's diameter and minimum cut, λ is an indicator of the graphs' underlying density. It is not a magic number when applying our triangle based algorithm in later sections. As proven in the Theorem 5.3.1, it is a local maximum. For example, in figure 5.1, subgraph $ABCDEF$ is a DN-graph of λ value 3. If we include one more vertex A', the λ value of the graph $A'ABCDEF$'s drops significantly to 0. Similarly, taking away any vertex, say A, leads to a lower λ value than 3.

DN-graph is designed to represent dense patterns, as it captures subgraphs with more internal associations. Appendix 5.3.2 depicts the relationship between λ and the clique size using a dynamically changing graph.

Besides the level of connectivity, a DN-graph also imposes restrictions on the minimal size of sharing neighborhood. This restriction is especially useful when predicting protein complexes via densely connected proteins inside a protein-protein interaction (PPI) graph. A protein complex's formation often serves to activate or inhibit one or more of the complex members[Wik06], in a PPI network, we can observe the phenomenon that members of a protein complex share (significantly many) neighbors. The DN-graph definition reconciles the sharing of neighborhood.

Based on DN-graph, this chapter discusses effective solutions towards mining

DN-graphs within a massive graph, Formally:

Definition 11. *DN-graph mining problem*

Given a graph $G(V, E)$, we want to find all DN-graphs $g(v, e, \lambda)$ in G.

Generally speaking, the level of interactions among entities determines the density of the substructures. From this point of view, it is not surprising to see that some patterns are transformable to others. For example, a DN-graph is a more general case of a closed clique (Recall that a clique is a fully connected graph, while the closed clique is the local maximal clique). In fact, a DN-graph is a relaxation of a clique, with less rigid size constraints. Lemma 5.3.1 states the relationship formally:

Lemma 5.3.1. *DN-Graph and Closed Clique*

A graph contains a closed clique of size d if and only if the graph contains a DN-graph g with $\lambda = d - 2$ and $|g| = d$.

The proof of above lemma follows two steps. First we prove the necessary condition. If a graph contains a DN-graph g with $\lambda = d-2$ and $|g| = d$, according to the definition of DN-graph, g has d vertices, which shares $(d - 2)$ common neighbors with every connected vertices in g. For any two distinct vertices v and u in g, if they are connected, they Both connect to every other vertices inside g. This holds for every vertex pair, which indicates that if g has an edge, it is a clique. While g is a DN-graph, it does not contain a proper super graph which is also a DN-graph, it thus does not have a proper super graph with $\lambda = d - 1$ and $|g| = d + 1$. It does not contain a clique of size $d + 1$.

The sufficient conditions proof is: if a graph contains a closed clique of size d (*), this clique has d vertices and each pair of vertices share $d - 2$ common neighbors. $\lambda = d - 2$ in this case according to λ definition. Being a closed clique, it does not have a proper super graph which is also a clique. This indicates that it does not have a proper super graph with $.\lambda \geq d - 1$ and $size = \lambda + 2$. Suppose there is a super graph with $lambda = d - 2 + \delta$ and $size = d + \epsilon$, there must be some vertices which are not inside g connecting to at least $d - 2 + \epsilon - \delta$ d-clique vertices, while $d - 2 + \epsilon - \delta \leq d$, indicating $\epsilon - \delta \leq 2$, which means $size \leq \lambda$. This is either impossible or the super graph is a clique, which contradicts with the assumption (*). So there is not possible such super graph exist. So the d-clique is an DN-graph.

With about, we finished the proof.

Using Lemma 5.3.1, we are able to reduce the close clique mining problem to DN-graph mining problem. The reduction signifies that DN-graph mining is NP-complete. A closed clique is the local maximal clique, where no proper super graph of it is also a clique. The problem of detecting cliques is a well known NP-complete problem, which is first discussed in the landmark paper [**?**]. As a clique possessing certain property (here, local maximality), a closed clique detection problem is also an NP problem. Prompted by this result, we seek to develop approximate solutions instead.

Like the closed clique mining problem, the computational bottleneck for DN-graph mining is on counting degrees within a subgraph. In fact, the counting of local degrees relies heavily on multiple joins of neighbors, which are computationally expensive. To avoid the complexity of multiple joins, we next introduce the concept of $\lambda(e)$.

5.3.4 DN-Graph and $\lambda(e)$

As discussed previously, the bottleneck of DN-graph mining is excessive number of neighborhood joins required. This is because we have to test combinatorial number of subgraphs for their λ value and most subgraphs tested are not DN-graphs.

Most of λ value testings however are unnecessary. Due to the local maximality feature of a DN-graph, it is impossible for any two different DN-graphs to share any common vertices or edges. Once we verify that a graph, g_{dn} is a DN-graph, we need not consider other subgraphs that intercept with g_{dn}. In fact, by computing the λ value of edges, we can locate DN-graphs. If we assign the λ value of g_{dn} as the density value of its edges, a DN-graph becomes a set of edges with local maximal λ.

Before explaining the process of locating DN-graph using edge density, let us first define edge density, $\lambda(e)$, formally:

Definition 12. $\lambda(e)$

Given a graph $G(V, E)$ and an edge $e \in E$, $\lambda(e)$ is the maximal $\lambda(G')$ value where $e \in E(G')$ and $G' \subseteq (G)$.

The value $\lambda(e)$ indicates quantitatively, the most prominent relationships between two linked vertices. With the definition of local density, we next prove that using $\lambda(e)$, we are able to find all DN-graphs.

Theorem 5.3.1. *Locating DN-Graph Using $\lambda(e)$*

A graph G' is a DN-graph if and only if

- *all edges e within G' have equal $\lambda(e)$ value, represented as λ_{max} and,*
- *for all $u \in N(G')$ and $v \in G'$, $\lambda(u, v) \leq \lambda_{max}$.*

Proof of Theorem 5.3.1 To prove the correctness of Theorem 5.3.1, we use the abstract graph in figure 5.4. The complete proof consists of two steps. Firstly, G' must exist. Secondly, G' must contain some DN-graph. To prove the existence of G', we construct G' using graph vertices/edges and their λ values. First pick a vertex v with $\lambda(v) \geq \lambda(u)$ for all $(u \in N(v))$. Denote $\lambda(v)$ as λ_{max}. By the definition of local λ value, $\lambda(v)$ participates in a connected graph G' with $\lambda(G') = \lambda_{max}$. From v, we find all its immediately connected neighbors that have $\lambda(u) = \lambda_{max}$. From each u, we find u's immediately connected neighbors with local λ value λ_{max}. This process propagates until no such neighbor exists. The collection of discovered vertices form a connected subgraph G' with λ value λ_{max}.

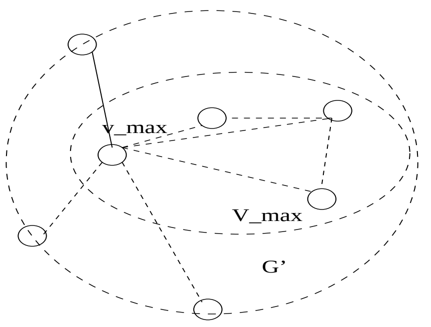


Fig. 5.4: Proof of Theorem 5.3.1

Next, we show that G' contains a DN-graph. By first part of the proof, G' contains all vertices and edges with λ value λ_{max}. For a vertex $v' \in G'$, it only can form DN-graph of $\lambda = \lambda_{max}$ with vertices inside G'. If denoting the minimal set of vertices from G' that form an DN-graph with v' as V_{min}, the subgraph $V_{min} \cup v'$ is also a DN-graph. This proves that a graph G' containing the set of vertices with $\lambda(v) = \lambda_{max} > \lambda(u)$ where $u \in N(G')$ must participate in a DN-graph. The condition that $\lambda(v) = \lambda_{max}$ and $\lambda_{max} > \lambda(u)$, where u is the neighbor vertices of G', means the graph G' contains vertices with local maximal λ value. Since graph G' is always a super graph of some DN-graph, If a solution can find G', the DN-graph can be located within G'.

With above two steps, we prove the correctness of Theorem 5.3.1.

Based on Theorem 5.3.1,we can locate the DN-graph by connecting edges with local maximal $\lambda(e)$.

Computing $\lambda(e)$ for all edges is however computationally prohibitive, as discussed in section 5.3. To facilitate approximation efficiently, we first find an upper bound value for $\lambda(e)$, the $\tilde{\lambda}(e)$, and then iteratively refine $\tilde{\lambda}(e)$ to capture the actual $\lambda(e)$ as accurately as possible.

The approximation is based on the fact that for an edge e, its $\lambda(e)$ value is upper bounded by the joint neighborhood size of the end vertices of e. This joint neighborhood size is in fact the number of triangles e participates in a graph. Thus we are inspired to use triangulation to approximate $\lambda(e)$ for every graph edge.

5.4 Local Triangulation and its Application in DN-Graph Mining

A triangle consists of a vertex triple (u, v, w) and three edges (u, v), (v, w) and (u, w). The problem of counting or listing all triangles within a graph is referred as **Graph Triangulation** in this paper:

Definition 13. *Graph Triangulation*

Given a graph $G(V, E)$, Graph Triangulation finds all vertex triples (u, v, w), where every vertex pair inside the triple are connected by an edge, denoted as $e(u, v)$, $e(v, w)$ and $e(u, w)$ respectively.

The joint neighborhood of an edge $e(u, v)$ upper-bounds $\lambda(e)$, while the number of triangles $e(u, v)$ participates in is equal to the joint neighborhood size. This indicates that graph triangulation provides an upper bound $\lambda(e)$ for every edge e.

Here we use $\tilde{\lambda}(u, v)$ to represent the current upper bound of edge (u, v). What's more, given a graph triangle, the $\tilde{\lambda}(u, v)$ can tighten the other two edges' density upper bound. The following proposition gives the relationship between an edge e's ($\tilde{\lambda}(e)$) and its neighbors':

Proposition 5.4.1. *Neighbor Bounding of* $\tilde{\lambda}(e)$

Inside a triangle (u, v, w)*, if* $\tilde{\lambda}(u, v) \leq min(\tilde{\lambda}(u, w), \tilde{\lambda}(v, w))$*, we say* w *supports* $\tilde{\lambda}(u, v)$*.* $\tilde{\lambda}(u, v)$ *is valid if and only if* $|\{w|w \text{ supports } \tilde{\lambda}(u, v)\}| \geq \tilde{\lambda}(u, v)$

The proof of necessary condition for proposition 5.4.1 follows the definition of the $\tilde{\lambda}$ value and is omitted for brevity. Now we prove the sufficient condition: If the number of supporting vertices is greater or equal to $\tilde{\lambda}(e)$, then $\tilde{\lambda}(e)$ is an upper bound for $\lambda(e)$. We prove this by contradiction. Suppose there are fewer than $\tilde{\lambda}(e)$ supporting vertices for $\tilde{\lambda}(e)$, according to the definition of $\lambda(e)$, $\lambda(e) < \tilde{\lambda}(e)$, which means $\tilde{\lambda}(e)$ is larger than $\lambda(e)$. In that case, $\tilde{\lambda}(e)$ is not a valid upper bound of $\lambda(e)$ This contradicts with earlier assumption. With the above reasoning, we complete the proof of proposition 5.4.1.

5.4.1 Triangulation Based DN Graph Mining

The elementary operation behind local triangulation is the joining of vertex neighborhoods. As studied in [Lat07], the performance of a local triangulation algorithm heavily depends on the order of those join operations. In fact, it is a necessary preprocessing step to sort vertices according to their degrees for effective triangulation. Below is the algorithm firstly proposed in [Lat07].

Algorithm 5.1 Local Triangulation Algorithm

```
Require: Graph $G(V, E)$
 1: $mk(e) = k(e) = TC(e) + 2, lbk(e) = 2$
 2: Order vertices and edges according to degrees
 3: for all dense vertex $v \in G$ do
 4:     Retrieve all $v$'s dense neighbors $u$
 5:     Joint $v$ and $u$'s neighborhoods to find triangle $< v, u, w >$
 6: end for
 7: for all sparse vertex $e(v, u) \in G$ do
 8:     Join $v$ and $u$'s neighbor lists to find triangles containing edge triangle $<$
        $v, u, w >$
 9: end for
10: return All triangles in $G$
```

Algorithm 5.1 separates the vertices in $G(V, E)$ into two classes: dense and sparse. Vertices with degree greater or equal to $\sqrt{|V|}$ are dense vertices, the remaining are sparse vertices. An edge with both end vertices being sparse vertices is a sparse edge. For a dense vertex v, the local triangulation algorithm perform a join on v's immediate neighbors and the neighborhood list v's neighbor, u. The size of the join set is the triangle count of edge (v, u). Similarly, for sparse edge (v', u'), algorithm join the neighborhood list of v' and u'

5.4.1.1 Generate Triangles to Refine Local Density

We adopt the graph triangulation algorithm in [Lat07]. The algorithm generates triangles systematically for each edge of the graph. The generation of the triangles is a sequence of join operations between the neighbors of two connected vertices. Based on a special order of joining operations, the triangles are generated in a streaming fashion. The DN-graph mining algorithm thus obtains the local density information gradually along the triangle streams. Based on proposition 5.4.1, we can use the number of triangles an edge participates in ($TC(e)$) as the initial upper bound of the $\lambda(e)$, the $\tilde{\lambda}(e)$. To give an even more accurate bound for $\lambda(e)$, the algorithm uses the density value of e's neighbors' to validate the current upper bound $\lambda(e)$. Figure 5.5 shows how this process works graphically.

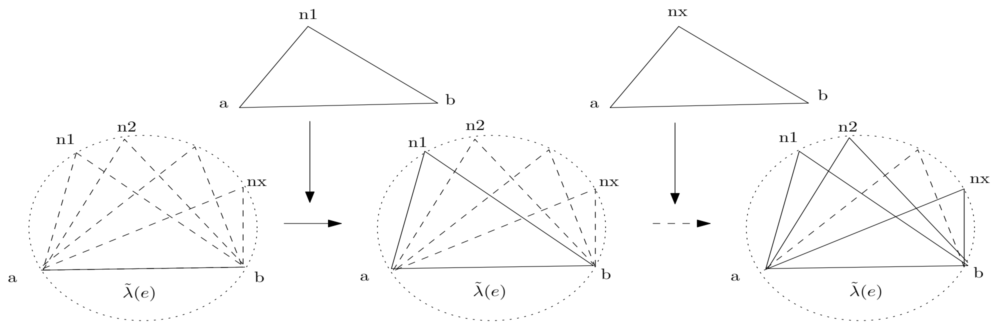


Fig. 5.5: Use Triangle to Refine $\tilde{\lambda}(e)$

In the first round of graph triangulation, we are aware of the triangular count of $e(a, b)$ (which is in fact $\tilde{\lambda}(e)$), and nothing about its neighbors. However, the triangular counts of the neighbors (a.k.a local density estimation) are available once the first round of graph triangulation is completed. To compute a more accurate $\tilde{\lambda}(e)$ for each edge, we will simply go through more rounds of triangulation and make use of the density information of the neighbors to further validate a new estimation of $\tilde{\lambda}(e)$ for each edge.

For a triangle $(a, b, n1)$, the algorithm checks whether the triangles $(a, b, n1)$ can possibly be a supporting evidence that edge $e(a, b)$ is in a DN-graph, with $\tilde{\lambda}(e)$. This is done by checking whether both the other two edges of triangle $(a, b, n1)$ (i.e. $e(a, n1)$ and $e(b, n1)$) have $\tilde{\lambda}$ greater or equal to $\tilde{\lambda}(e)$. If this is the case, this means that $n1$ is such a supporting vertex.

The triangle is then represented as a solid line indicating that $e(a, b)$ finds a new supporting vertex $n1$ in DN-graph with $\tilde{\lambda}(e)$. As new triangles approach, the algorithm counts the number of supporting vertices for edge (a, b) to form DN-graph, with current value of $\tilde{\lambda}(e)$. After one pass of all triangles, the number of vertices that support each edge's density upper bound $\tilde{\lambda}(e)$ are available for further computation.

Algorithm 5.2 Triangulation based DN-Graph Mining

```
Require: Graph $G(V, E)$
1: Triangles = Triangulation(G), $k(e)$=Triangle_count(e)
2: while converge AND iteration!=MAX_ITR do
3:    $sc = 0$, converge=TRUE
4:    for all Triangles $(a, b, c) \in G$ do
5:       Increment corresponding $sc(e)$ if $e$ is supported
6:    end for
7:    for all edges $e \in G$ do
8:       if $(sc(e) < \tilde{\lambda}(e))$ then
9:          Find next possible value $\tilde{\lambda}(e)$ for $e$
10:         converge = FALSE
11:      end if
12:   end for
13:   Increment iteration by 1
14: end while
15: return $\tilde{\lambda}(e)$ for each $e \in E$
```

With the supporting neighbors' information, the algorithm is able to determine the upper bound of λ for each graph edge (the upper bound is denoted as $\tilde{\lambda}(e)$). If sufficient supporting vertices are found for $\tilde{\lambda}(e)$ for an edge $e(a, b)$, $\tilde{\lambda}(e)$ is a valid upper bound of $e(a, b)$'s λ value. If there is not enough supporting vertices for $e(a, b)$, the algorithm finds the next possible $\tilde{\lambda}(e)$ value and tests it in the next round of triangulation. The algorithmic description is given in Algorithm 5.2. Within the algorithm, $sc(e)$ records the number of vertices supporting current $\tilde{\lambda}(e)$ value.

5.4.1.2 $\lambda(e)$ Bounding Choice

We can derive two variants of DN-graph mining algorithms from Algorithm 1, namely algorithms TriDN and BiTriDN. The two algorithms have different ways to decide the next possible $\tilde{\lambda}(e)$ value. The first variant, called **Tri**DN, decreases $\tilde{\lambda}(e)$ by one (Line 9 in Algorithm 5.2 becomes $\tilde{\lambda}(e) = \tilde{\lambda}(e) - 1$), if current $\tilde{\lambda}(e)$ cannot obtain sufficient supporting vertices count. This strategy is useful when the triangle counts are close to the actual $\lambda(e)$ values (qualitatively, when $|TC(e) + 2 - \lambda(e)| \leq log\lambda(e)$).

When the triangulation results are far above actual $\lambda(e)$, we can employ the second variant, called **BiTri**DN, which adopts a binary search strategy for the next possible value of $DN(e)$. BiTriDN requires additional information of possible $DN(e)$'s range. We use two numbers $lbk(e)$ and $\tilde{\lambda}(e)$ to record the lower bound and upper bound of $\lambda(e)$ value, and $mk(e)$ denotes the medium of range $[lbk(e), \tilde{\lambda}(e)]$. For completeness, we rewrite Line 7 onwards in Algorithm 5.2. BiTriDN has the advantage of fast convergence if the graph to be mined has many high degree vertices (qualitatively, when $|TC(e) + 2 - \lambda(e)| \geq log\lambda(e)$).

Algorithm 5.3 Binary DN-Graph Mining Variance "**BiTri**DN"

```
Require: Graph $G(V, E)$
 1: $mk(e) = k(e) = TC(e) + 2$, $lbk(e) = 2$
 2: Get support count $sc_{mk}(e)$ for all edges' $\tilde{\lambda}(e)$ {This part is the same as in
    Algorithm 5.2}
 3: for all edge $e \in G$ do
 4:     if ($sc_{mk}(e) < mk(e)$ AND $lbk(e) < \tilde{\lambda}(e)$) then
 5:         $\tilde{\lambda}(e) = mk(e) - 1$, converge = FALSE
 6:     else
 7:         $lbk(e) = mk(e)$
 8:     end if
 9:     $mk(e) = \frac{\tilde{\lambda}(e)+lbk(e)}{2}$
10: end for
11: return $\tilde{\lambda}(e)$ for each $e \in E$
```

Correctness of $\lambda(e)$ Bounding Choices

If we denote the actual local λ value for an edge as $\lambda(e)$, and exact supportive neighbor count as sc. The upper bound of λ value is denoted as $\tilde{\lambda}(e)$ and the supportive neighbor count of $\tilde{\lambda}(e)$ is denoted as $sc_k(e)$. To prove the correctness of algorithm DN-graph triangulation mining in Algorithm 5.2, we need to proof that 1). $\tilde{\lambda}(e)$ is always an upper bound of $\lambda(e)$. 2). $\tilde{\lambda}(e)$ converges.

Proof. 1) For the first bounding choice, At the beginning of the Algorithm 5.2, $\tilde{\lambda}(e)$ is equal to triangle count for e, $TC(e)$. since $\tilde{\lambda}(e) \geq \lambda(e)$, if the algorithm stops now, the upperbound invariance holds. Suppose the invariance holds for iteration $i > 0$, $\tilde{\lambda}(e) \geq \lambda(e)$, at iteration $i+1$, $\tilde{\lambda}(e)$ is updated to $\tilde{\lambda}(e) - 1$ when this condition holds: the number of neighbors having λ values greater or equal to $\tilde{\lambda}(e)$ is less than $\tilde{\lambda}(e)$. This condition uses neighborhood vertices' k values to verify if current iteration's $\tilde{\lambda}(e)$ value is held. Since the neighbor's $\tilde{\lambda}$ values upperbounds their actual λ values, the number of qualified candidates $sc(e) \geq sc_k(e)$. Thus when $sc_k(e)$ is less than $\tilde{\lambda}(e)$, $\tilde{\lambda}(e)$ is definitely greater than the real $\lambda(e)$ value (as in 5.4.1). In that case, $\tilde{\lambda}(e)$ is reduced by 1. And the new $\tilde{\lambda}(e)$ value is still an upperbound. The upper bounds invariance is proven to hold. 2) For the second bounding choice (as in Algorithm 5.3), the proof of upper bound invariance follows with the exceptions that this bounding choices test on the median of possible λ range instead of $\tilde{\lambda}(e) - 1$.

The convergence of $\tilde{\lambda}(e)$ is due to monotonic decreasing of $\tilde{\lambda}(e)$ values. The algorithm initializes $\tilde{\lambda}(e)$ as triangle counts. This is an upper bound of $\lambda(e)$. After that, algorithm only decreases $\tilde{\lambda}(e)$. As $\lambda_a(e)$ always upperbounds actual value $\lambda(e)$, $\lambda_a(e)$ value will converge. □

5.4.2 Triangulation based DN-Graph Mining Algorithm Complexity Analysis

The triangulation algorithm (in Appendix 5.1) sorts vertex and adjacency list into descending order of degrees. The operations require $O(|V|log|V|)$ time complexity. After that, it counts triangles inside the graphs for each vertex. To count triangles, algorithm separates vertices into dense vertices and sparse ones according to vertices' degrees. For dense vertex, the algorithm lists the number of triangles $|V|$ participating in $O(|E|)$ time. The total complexity for counting dense vertices is $O(|V||E|)$. For sparse vertices, the algorithm counts sparse edges that intersects with sparse vertices. For a sparse vertex/edge, its neighborhood size is at most constant (say S). The counting over sparse edges requires $O(S|E|)$ time. If setting $S = \sqrt{|}E|$, taking into consideration of the complexity of counting on dense vertices, The time complexity for triangle counting procedure is $O(|E|^{\frac{3}{2}})$.

The algorithm TriDN in Algorithm 5.2 iterates on all triangles that form the graph. Each iteration also requires $O(|E|^{\frac{3}{2}})$ time. For a fixed number of iteration k, the algorithm needs $O(k|E|^{\frac{3}{2}})$ time in total. If insisting on convergence, the algorithm may need up to $O(k|V||E|^{\frac{3}{2}})$. As we may need to test local λ value $\lambda(e)$ from $v - 2$ down to 3. The iterative version of the algorithm for λ mining reduces time complexity to $O(klog|V||E|^{\frac{3}{2}})$ since it employs the binary search paradigm to test possible $\lambda(e)$ for every e. The space complexity of both DN-graph mining algorithms are similar. The triangulation stage requires $O(|V|log|V| + |E|)$ while the iteration process requires $O(|E|)$.

When the triangulation results are far above the actual $\lambda(e)$ value, we can employ the second variant, called **BiTri**DN, which adopts a binary search strategy for the next possible value of $DN(e)$. BiTriDN requires additional information of possible $DN(e)$'s range. We use two numbers $lbk(e)$ and $\tilde{\lambda}(e)$ to record the lower bound and upper bound of $\lambda(e)$ value, and $mk(e)$ denotes the medium of range [$lbk(e)$, $\tilde{\lambda}(e)$]. For completeness, we rewrite Line 7 onwards in Algorithm 5.2. BiTriDN has the advantage of fast convergence if the graph to be mined has many high degree vertices (qualitatively, when $|TC(e) + 2 - \lambda(e)| \geq log\lambda(e)$).

5.5 Extension of DN Graph Mining to Semi-Streaming Graph

The semi-streaming graph model assumes the vertices of the graph can be fitted into main memory, and the interactions among vertices are stored in an ordered manner within the secondary storage. While this assumption may not hold for arbitrarily large graphs, we can still handle up to Giga scale vertices (assume $|V|$ vertices require $|V|log|V|$ bits storage) with today's main memory capacities. Following the nature of physical storage devices, our streaming model assumes random access in primary storage (i.e. memory) and only sequential access in secondary storage. In the secondary storage, graph interactions are stored in the form of adjacency list. As a feasible solution towards a streaming graph $G(V, E)$, it should not exceed $log|V|$ scans of G's adjacency list.

In the semi-streaming graph setting, the exact triangulation algorithm proposed in [Lat07] cannot be directly applied in the DN-graph mining solutions. The information of the neighbors are stored in secondary storage and may not be immediately available when the algorithm retrieves it.

In view of above difficulty, our streaming solution first performs a semi-streaming triangulation, followed by the complete DN-graph mining solution in semi-streaming setting.

The neighborhoods join operations are in fact the process of determining the similarity between two sets. The most well-adapted measurement for set similarity is Jaccard coefficient. For two sets A and B, Jaccard coefficient is calculated as $J(A, B) = \frac{|A \cap B|}{A \cup B}$.

In the semi-streaming graph setting, it is however expensive to calculated Jaccard coefficient between two neighborhoods. Since the operation of set joining requires expensive pre-processing of sets such as sorting or heap building.

In view of above difficulty, we use the property of min-wise independent set to approximate Jaccard coefficient. When dealing with large sets, min-wise independent property approximate set intersection size using sequential scan only.

Suppose A and B are defined on the set universe X, and π is a permutation over universe X, the min-wise independent property states: If $\pi[X]$ is a uniformly chosen random permutation over X, and $W \subset [X]$ is any subset over the universe, and $\pi[W]$ is the projection of W by permutation π, then the probability that two subsets' minimal projected images are equal is the same as the Jaccard coefficient. Formally, $P[min(\pi[A]) == min(\pi[B])] = J(A, B)$.

5.5.1 an Estimated Triangulation Algorithm

[BBCG08] proposes a streaming local triangle counting algorithm based on min-wise independent property:

Algorithm 5.4 Streaming Triangulation Algorithm

```
Require: Graph $G(V,E)$, $r$ : # of scans of graph links, $k$ : # of bits for hash
    values
 1: $mk(e) = k(e) = TC(e) + 2, lbk(e) = 2$
 2: $Y = 0, min(V) = 0$
 3: for $s = 1$ TO $r$ do
 4:    Hash every vertex label $h_s(v)$ to any random $k$ bits
 5:    for all vertex $v \in V$ and its neighbor $u$ do
 6:       min(v) = min( min(v),$h_s(u)$ )
 7:    end for
 8:    for all vertex $v \in V$ and its neighbor $u$ do
 9:       if (min(u)==min(v)) then
10:          Increment $Y(u,v)$ by 1
11:       end if
12:    end for
13: end for
14: for all every vertex $v$ in $G$ do
15:    $TC(v) = \sum_{u \in N(v)} \frac{Y(v,u)}{Y(v,u)+r}(|N(v)| + |N(u)|)$
16: end for
17: return All triangles in $G$
```

The basic idea of streaming triangulation algorithm in Algorithm 5.4[BBCG08] generates r times permutation over vertices set V. For each permutation, algorithm records every vertex' minimal neighbors under this permutation. After getting the minimal neighbors for each vertex, the algorithm scan the graph once to compare if the minimal neighbors of two immediately connected vertices are equal. If they are equal, algorithm increments count Y for the connected vertices. The estimator for vertex triangle count is

$$\sum_{u \in N(v)} \frac{Y(v,u)}{Y(v,u)+r}(|N(v)|+|N(u)|)[\text{BBCG08}]$$

. This estimator is derived from min-wise independent property. We use limited number of permutation to estimate all permutations over graph vertices.

5.5.2 Streaming DN-Graph Mining Algorithm Detail

The algorithm 5.4 estimates local triangulation using edge scans. It forms the first step of algorithm StreamDN. The next step is to calculate each edge's λ value using only edge scans [1]. StreamDN, as presented in Algorithm 5.5, adopts the bounding process as algorithm BiTriDN. That is:

Algorithm 5.5 Streaming DN-Graph Mining Algorithm "StreamDN"

```
Require: Graph $G(V,E)$, $r$ : # of scans of graph links $k$ : # of bits for hash values
 1: $mk(e) = \tilde{\lambda}(e) = TC(e)$, $lbk(e) = 0$
 2: Triangulation and store triangle count $TC(v,u)$ for all $e \in E$ as in algorithm 5.4 in appendix.
 3: while !converge AND iteration!=MAX_ITR do
 4:     $sc_k = 0$ $ubk(e) = \tilde{\lambda}(e) = TC(e)$, $lbk(e) = 0$
 5:     for all edge $(u,v) \in G$ do
 6:         $sc_k(u,v)$=number of $u$'s neighbor with $\tilde{\lambda}(u,v)$
 7:         Bound $\tilde{\lambda}(u,v)$ using $ubk(u,v)$/$lbk(u,v)$/$sc_k(u,v)$ {the same as Algorithm 5.3}
 8:     end for
 9: end while
10: return $\tilde{\lambda}(e)$ value for every graph edge $e$
```

[1] For brevity, in following parts of the paper, we use streaming DN-graph mining algorithm instead of explicitly stating "semi-streaming"

The only difference between the streaming version of the algorithm and BiTriDN is when counting the supporting vertices. In StreamDN, we can only access the graph edges sequentially. In view of the restriction, proposition 5.4.1 is relaxed to as follows:

Proposition 5.5.1. ***Relaxed*** *Neighbor Bounding of* $\lambda(e)$

Given a graph edge $e(u, v)$ *and the joint neighbor set* $N_\cap(u, v)$*, we say a vertex* $w \in N_\cap(u, v)$ *is a supporting vertex of* $\tilde{\lambda}(e)$ ***if*** $\lambda(u, w) \geq \tilde{\lambda}(e)$*. An integer* k *is a valid upper bound of* $\tilde{\lambda}(e)$ *if and only if there are at least* k *of such supporting vertices in* $N_\cap(u, v)$

The proof for proposition 5.5.1 is omitted for brevity.

5.5.3 Error-Bound on Streaming DN-Graph Mining

As mentioned in the previous subsections, there may be error during the limited number of permutations when applying min-wise independent set property. The error, however, is bounded. If we denote the joint size as $X = |A \cap B|$ and the estimated value $\overline{X} = TC(A, B)$, the error bound is:

$$P[|\overline{X} - X| > \epsilon X] \geq 2e^{-\frac{\epsilon^2}{3} rJ(A,B)} + \frac{m|A \cup B|}{2^k - 1} \text{[BBCG08]}$$

As mentioned in the previous subsections, the number of permutations adopted determines the estimation accuracy of min-wise independent property. The error, however, is bounded. If we denote the joint size as $X = |A \cap B|$ and the estimated value $\overline{X} = TC(A, B)$, the error bound is:

$$P[|\overline{X} - X| > \epsilon X] \geq 2e^{-\frac{\epsilon^2}{3} rJ(A,B)} + \frac{m|A \cup B|}{2^k - 1}\text{[BBCG08]}$$

5.5.4 Complexity Analysis for Streaming DN-Graph Mining

Since the algorithm Streaming DN-graph mining in Algorithm 5.5 first performs semi-streaming triangulation following the idea proposed in [BBCG08]. While the semi-streaming triangulation scans the graph r passes to apply min-wise independent set principle and an additional set to calculate the estimator of triangle counts, its complexity is of $O(r|E|)$. The following steps of streaming DN-graph mining requires $O(|E|)$ time for every iteration. For a fixed number of iteration w, the algorithm needs $O(w|E|)$ time. In summary, the time complexity of streaming DN-graph mining is of $O(k|E|)$, where k is a constant. In fact, the triangle based approach can also be applied to dynamic graphs.

5.6 Dynamic DN-Graph Mining

This section discusses solution towards DN-graph mining for graphs whose topology does not change over time. This solution can be extended to graphs with dynamic topology with minimal modification.

Recall that the triangulation based DN-graph mining solution consists of two stages. In the first stage, algorithm in Algorithm 5.2 performs local triangulation on the whole graph. The next step is to iterate on each discovered triangle. For each triangle, the algorithm uses it to verify whether it can support its edges' $\lambda_a(e)$ value. After scanning all triangle once, the $\lambda_a(e)$ value for each edge are updated accordingly.

A dynamic graph $\mathcal{G}(\mathcal{V}, \mathcal{E}) = \{G^t(V^t, E^t)\}$ changes its topology over time. Such dynamics can also be modelled as the emerging and disappearing of triangles inside $\mathcal{G}$. At each discrete time t, the instance of a streamed graph is the set of vertices and edges presented at t, we use $G^t(V^t, E^t)$ to represent it. Without loss of generally, this chapter only concerns the addition of edges. When a new edge e appears between vertex a and b, this edge's initial $\lambda_a(e)$ is set to be the size of $N(a) \cap N(b)$, while edge $\lambda_a(a, n)$ and $\lambda_a(b, n)$ increase by 1 if they share neighbor vertex n before new edge comes. After the process of adjusting $\lambda_a(e)$ values for dynamically affected edges, we then iterate on the new set of triangles following the same way as in algorithm in Algorithm 5.2.

5.6.1 Complexity for Dynamic DN-Graph Mining

Denote the total number of edges of a dynamic graph $\mathcal{G}$ as $|E^*|$. Followed previous discussion, we only process each edge when it first appears in the dynamic graph or its neighborhood information changes. If the graph is a sparse graph, the neighborhood size of any vertex can be treated as a constant. Thus in sparse graph, the complexity for dynamic DN-graph mining is of $O(k|E^*|)$ where k is the number of iterations. Later in experimental section, we use real data to evaluate the performance of the dynamic version of the algorithm and the empirical time efficiency is always much smaller than the theoretical bound. The space complexity is $O(E^*)$ as we need to store $\tilde{\lambda}$ information for each edge appeared.

5.7 Experimental Study

In this section, we study the performance of the DN-graph mining algorithms. Experimental data come from both theoretically proven data generators ([BC96]), as well as domain datasets. All the experiments are conducted on a workstation with a Quad-Core AMD Opteron(tm) processor 8356, 128GB RAM and 700GB hard disk. The operating system is Windows server 2003, Enterprize x64 edition.

Synthetic Graph Generators (G_{EC}). We use the *clique hiding graph generator* developed by M. Brockington and J. Culberson [BC96]. This graph generator randomly embeds a fixed size clique (c) into a graph of ($|V|$) vertices. The graph density, p, is calculated as $p = \frac{2|E|}{|V|(|V|-1)}$. The resulting graphs are random graphs with one known fixed size clique embedded. Table 5.2 summarizes the key parameters, with the default values highlighted in **bold**.

Domain Graph Datasets We also employ 3 real life datasets in our study.

Parameters	Experimental Range
c: clique size	[20, **40**, 60, 80, 100]
$\|V\|$: # of vertices	[1000,2000, **3000**, 4000, 5000]
p: edge density(%)	[4, 8, **12**, 16, 20]

Tab. 5.2: DN-Graph Mining Experiment Parameter Table

These datasets are either collected by domain experts or extracted from well-known public databases. 1) Protein Protein Interaction (PPI) dataset: This dataset [XSe02] contains 17203 interactions among 4930 proteins.2) Netflix dataset: It is compiled from Netflix raw data consisting of 480,000 customers and 17,000 movies records [net]. 3) Flickr dataset: This dataset is derived from the well-known photo sharing network Flickr with 1,715,255 vertices and 22,613,982 edges. Each vertex represents a person.

5.7.1 Performance Evaluation

The first set of experiments evaluate the accuracy of DN-Graph algorithms on the synthetic data generated by G_{EC}. We focused on two algorithms - TriDN and BiTriDN. We compared the results and observed similar behaviors between the two algorithms. Due to space limitation, we only present the results of algorithm BiTriDN.

There are three groups of experiments, each of which fixes a G_{EC} parameter to the default value. Group 1: When we fixed $|V| = 300$, Figure 5.6 and figure 5.6

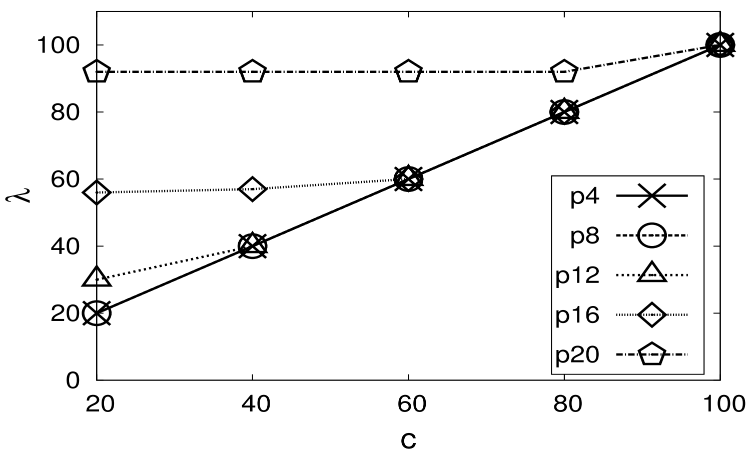


Fig. 5.6: Fix $|V| = 3000$, Vary c

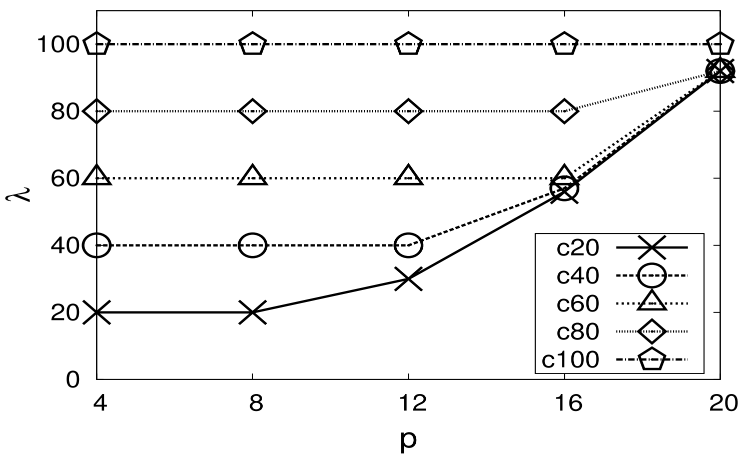


Fig. 5.7: Fix $|V| = 3000$, Vary p

show the calculated λ values of different datasets. From figure 5.6, the algorithm accurately reports the DN-graph size as c, when the embedded clique is in fact the dense area of the dataset. When the graph is denser ($p \geq 0.12$), the clique becomes a less dense area. In these datasets, BiTriDN reports higher $\tilde{\lambda}$ value (up to $\tilde{\lambda} = 90$). We examined the dataset and verified that BiTriDN did find DN-graph with higher $\tilde{\lambda}$ value ($> c$), and confirmed the correctness of BiTriDN's results. Similarly, the results in figure 5.7 shows the $\tilde{\lambda}$ value over different densities, which once more confirms BiTriDN's accuracy. Group 2: When we fixed $p = 12\%$, the results in figures 5.8 and 5.9 show that BiTriDN identifies the λ value accurately. Similar observations are made for experiments in Group 3 when we fixed $c = 40$ (see figures 5.10 and 5.11).

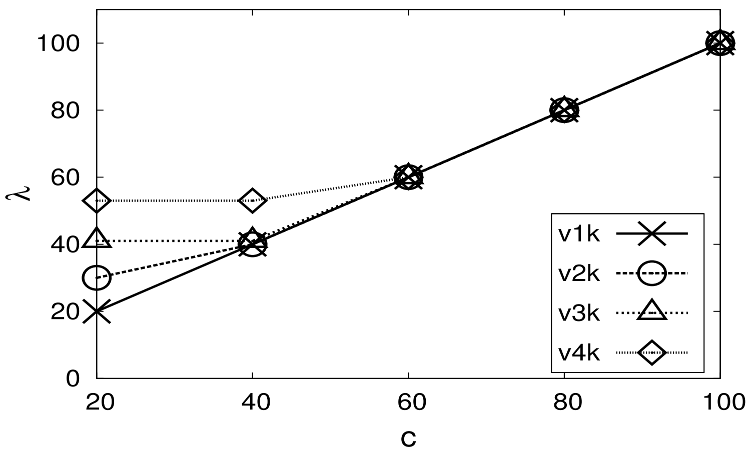


Fig. 5.8: Fix $p = 12\%$, Vary c

Convergence of DN-Graph Mining Algorithms

In this set of experiments, we compare the pace of convergence between two algorithms: TriDN and BiTriDN. The synthetic datasets are generated from G_{EC} as well. The maximal iteration is set to be 40 rounds to avoid pro-long running of the experiments. Plots in figures 5.12 and 5.13 show the number of iterations to reach convergence with different graph density p and graph size $|V|$. The results from both plots show that algorithm BiTriDN can converge within 10 to 25 round-

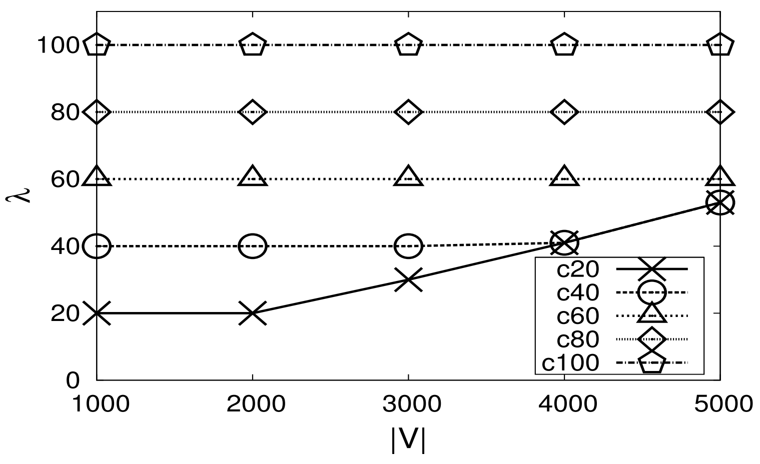


Fig. 5.9: Fix $p = 12\%$, Vary $|V|$

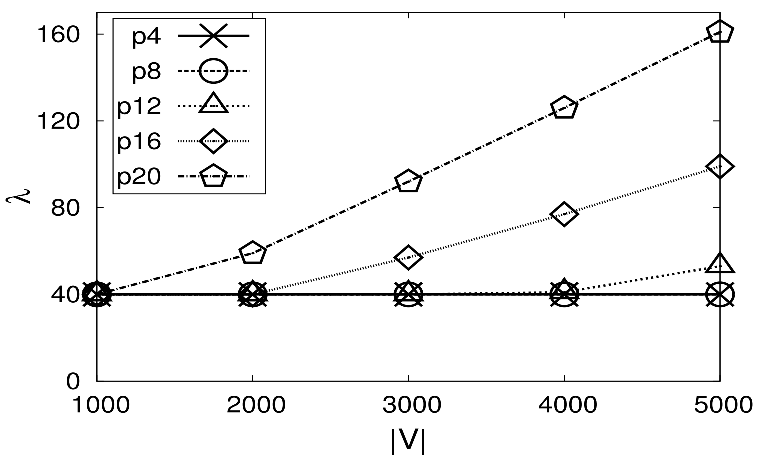


Fig. 5.10: Fix $c = 40$, Vary $|V|$

s on most parameter settings. However, in most of the experiments, TriDN has not reached convergence after the preset maximal iteration. This provides strong support on the claim that BiTriDN converges significantly faster than TriDN.

Time Performance in Memory

In this study, we evaluate the efficiency (i.e., running time) of the two algo-

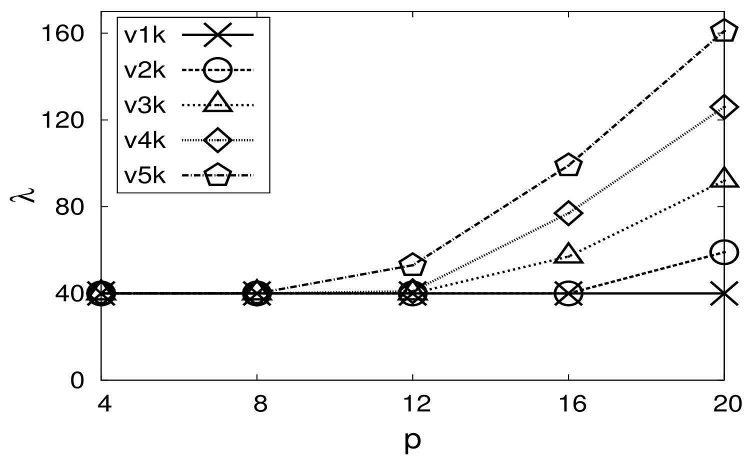


Fig. 5.11: Fix $c = 40$, Vary p

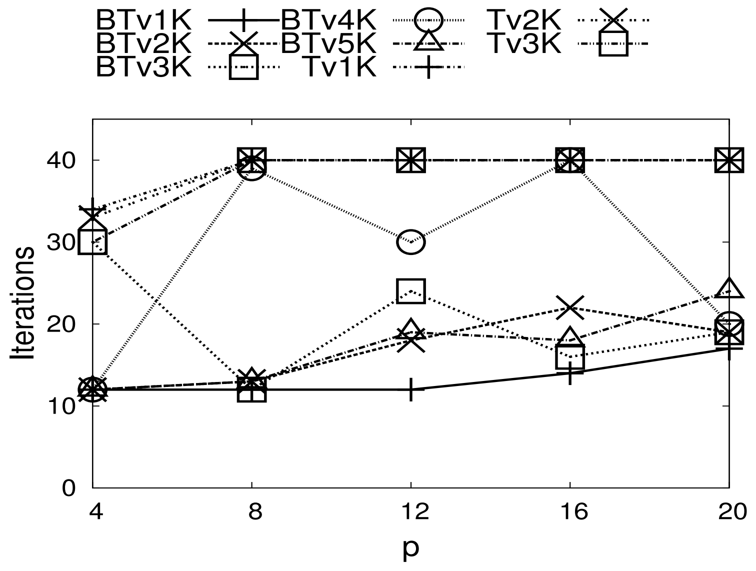


Fig. 5.12: Convergence: Vary p, fixed c

rithms TriDN and BiTriDN over the synthetic data generated by G_{EC}. For a fixed parameter setting, the two algorithms converge at different iteration. To remove the effect of different convergence speed towards time performance, all time are measured for only 1 iteration in this study. Both algorithms have almost same

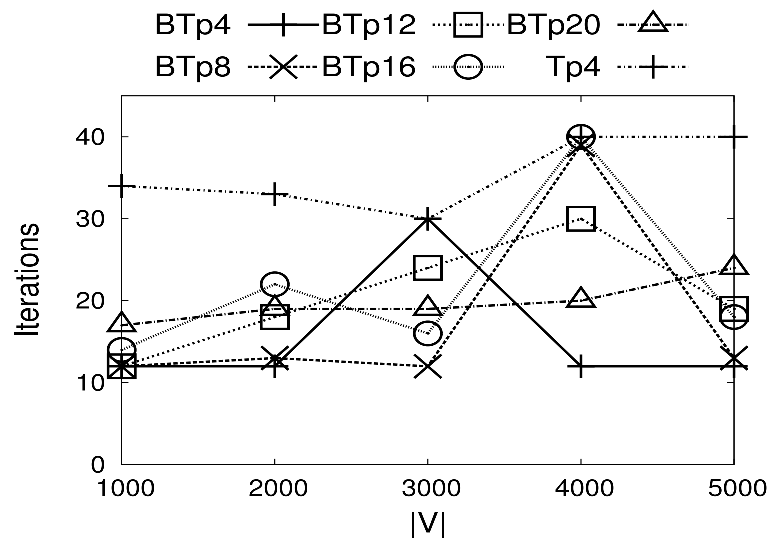


Fig. 5.13: Convergence: Vary $|V|$, fixed $c = 40$

behavior for 1 iteration.

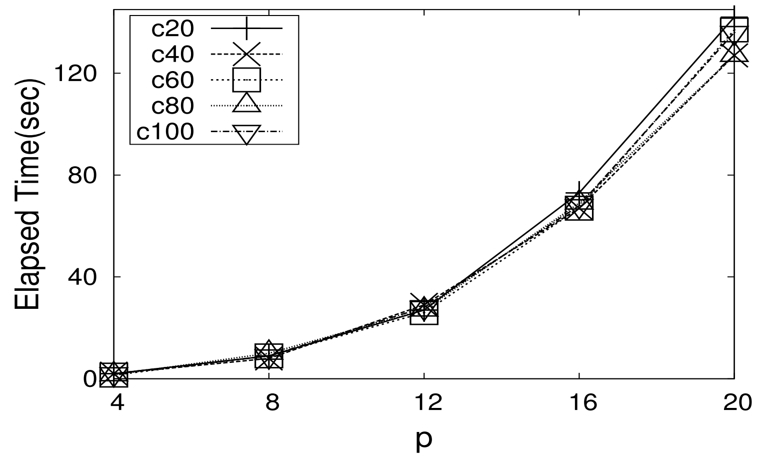


Fig. 5.14: BiTriDN One Iteration: $|V| = 3000$ Vary p

Figure 5.14 and 5.15 show the running time when $|V|$ is fixed. The results match the complexity analysis in section 5.4.2. The effects of edges distribution change are shown in figure 5.14 and 5.15. The synthetic graph generator G_{EC}

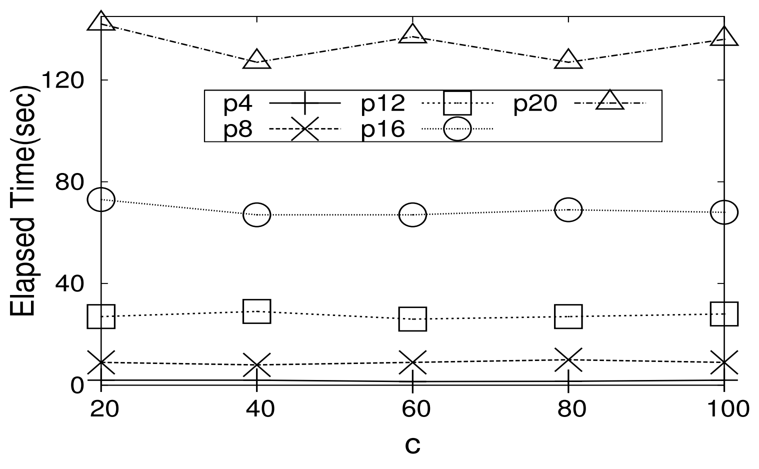


Fig. 5.15: BiTriDN One Iteration: $|V| = 3000$, Vary c

varies the edge distribution by varying the embedded clique size c. Experiments on these data always indicate that only $|V|$ and p affect the running time. Figure 5.16 and 5.17 present the effect of different graph size $|E|$ over the running time. The trend over time roughly follows complexity $O(E^{\frac{3}{2}})$.

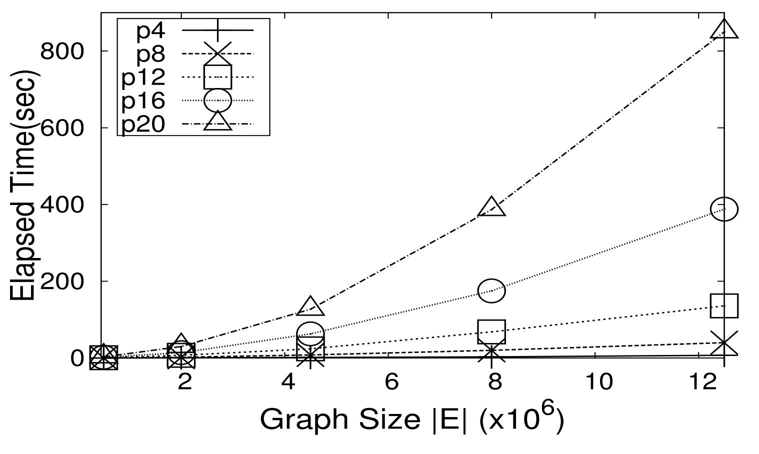


Fig. 5.16: BiTriDN One Iteration: Vary $|V|$, $c = 40$

Efficiency Improvement over Algorithm CSV

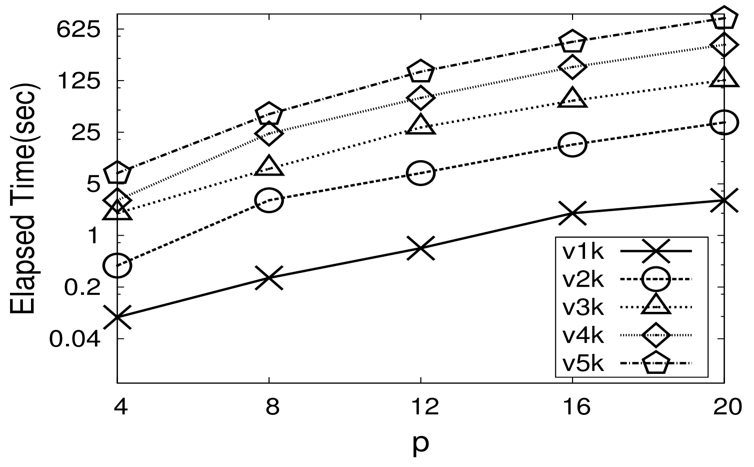


Fig. 5.17: BiTri$DN$ One Iteration: Vary p, fix $c = 40$

Since both CSV algorithm presented in previous chapter (chapter 4) and the triangle based $DN$-graph mining algorithms aims at discovering dense subgraphs out of large graphs, we conducts a set of experiments to compare the time efficiency between algorithms CSV and BiTri$DN$. The results are presented in figure 5.18. Since CSV is an in-memory algorithm, we only compare it with BiTri$DN$, the in-memory representative of $DN$-graph mining algorithm family.

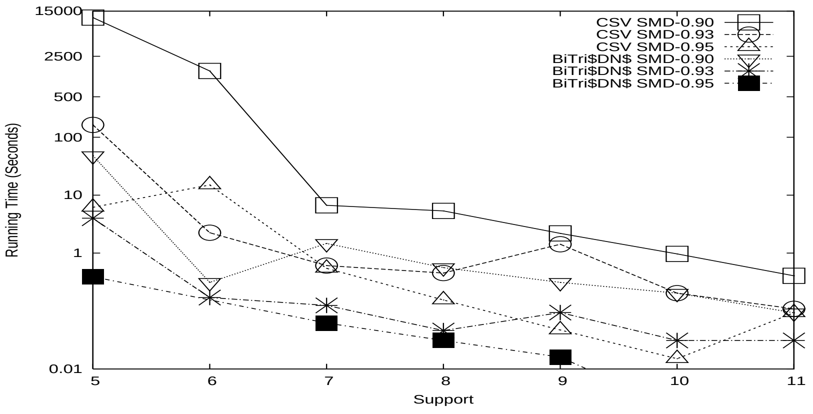


Fig. 5.18: Efficiency BiTri$DN$ vs CSV

To compare the time efficiency between CSV and BiTriDN algorithms, we run both algorithms on the stock market data sets (the size of the dataset is listed in table 4.3) with support 5 to 11. For fair comparison, we set the maximum number of iterations of BiTriDN graph to be 20, and most stock market graphs can reach convergence under this max iteration. The mapping dimension for CSV is set to 4. From figure 5.18, we can see that BiTriDN algorithm out-performs CSV more when the graph size increases. Taking the stock market graph with correlation 90 and support 5 as an example, CSV takes 11630 seconds to produce results, while BiTriDN only requires 48 seconds. The significant reducing in processing time is mainly because BiTriDN's adoption of triangulation based approximation of neighborhood size. With triangulation, BiTriDN's complexity in neighborhood approximation is $O(|E|^{\frac{3}{2}})$. Compared with the CSV's complexity in neighborhood bounding $O(|V|^2 \log |V| 2^d)$ (d is the mapping dimension), the gain is in the order of magnitude.

Recursively Applying Triangulation for More Accurate results Since DN-graph mining algorithms always discovers super graphs of actual DN-graph, if we apply these algorithm recursively on discovered super graphs, we should be able to local DN-graph more accurately and with less round of iteration.

To verify above assertion, we apply BiTriDN algorithm on a set of stock market graphs twice. After the first run, we identify huge dense subgraphs. We then abstract one with largest density and use BiTriDN to mining dense subgraphs out of it. The results are shown in figure 5.19.

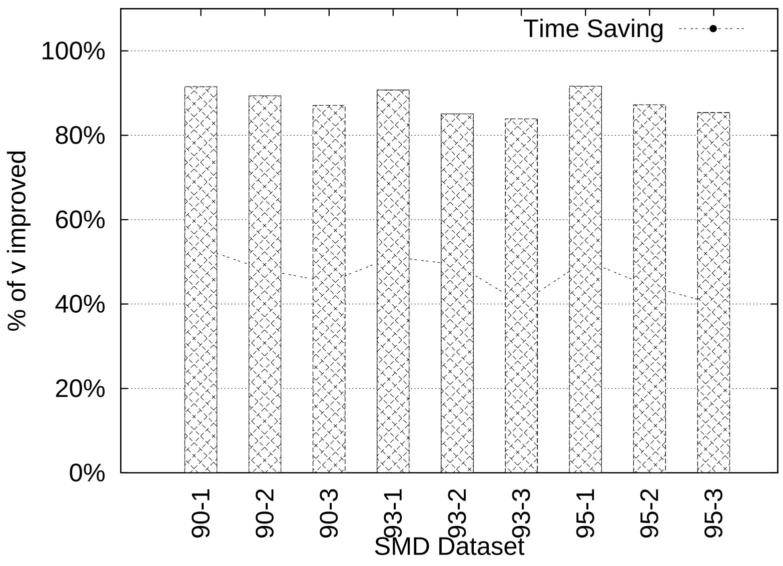


Fig. 5.19: Improvement by Recursively Applying Triangulation

Figure 5.19 shows the improvement of recursive BiTriDN over its non-recursive application. The bars show the that among the 9 graphs we tested, recursive BiTriDN can improve more than 85% vertices' density. The line in figure 5.19 indicates the time savings of recursive BiTriDN, for the 9 stock market graphs, the gain of running time is around 50%. From this experiment, we conclude that recursively applying BiTriDN can produce more accurate result quickly for a specific subgraph.

Memory Usage

We also monitor the peak memory usage of both TriDN and BiTriDN on the synthetic data. Figure 5.20 shows that both algorithms increase their memory usage when the graph size/density increase. Meanwhile, the results tell us that the memory usage of TriDN is always slightly less than that of BiTriDN under the same parameter setting. This is because in BiTriDN, additional memory is used to store both the upper bound and lower bound of the λ value.

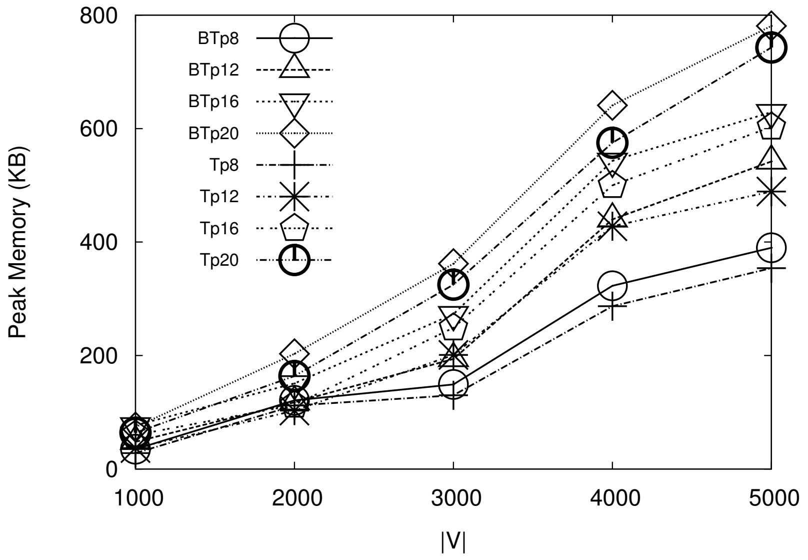


Fig. 5.20: Memory Usage of TriDN and BiTriDN

Test on Very Large Graph

In this experiment, we applied BiTriDN on the Flickr dataset. The running time per iteration is between 55 minutes to 1 hour. The stable memory usage is less than 1G. The program converges after 66 iterations. Each iteration's λ values are recorded for each vertex. When algorithm converges, the largest DN-graph's highest λ is 278. We note that at the 35th iteration, the largest λ value already reaches 279. Figure 5.21 plots the trend of maximal vertex λ value's change. From this experiment, the DN-graph mining results have high availability as the results are updated at every iteration. What's more, if allowing small errors, the program converges very fast. These fast convergence feature is observed for all real datasets we tested on. We did not reports all results due to space limitation.

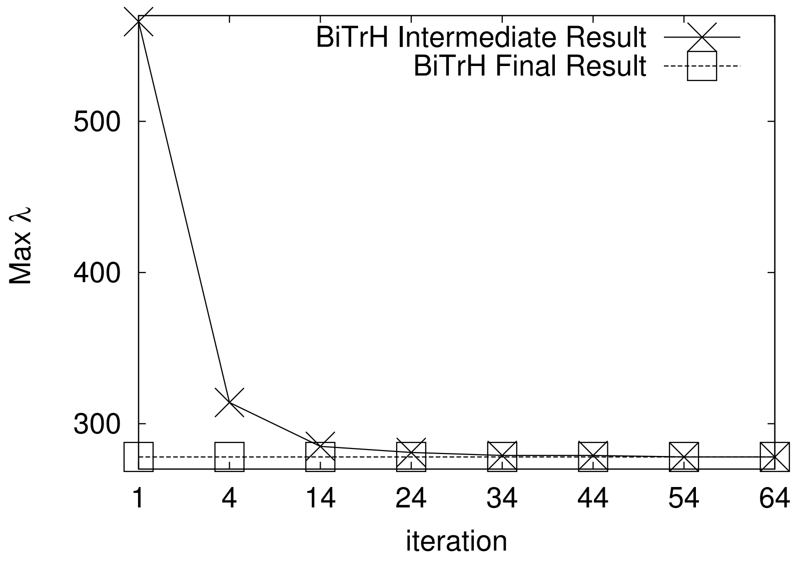


Fig. 5.21: Performance on Flickr Dataset: Convergence

StreamDN Performance on Flickr Dataset

In this set of experiments, the StreamDN algorithm is ran on the Flickr dataset. The implementation mimics the behavior of disk scan on main memory as our experiment machine has large enough memory. The results in figure 5.22 shows StreamDN over-estimates with respect to BiTriDN algorithm's results by 72% during the first 66 scans of the whole Flickr dataset. With the analysis of StreamDN's running time complexity, we confirmed that the triangulation based DN-graph mining algorithms can handle streaming setting with reasonable accuracy.

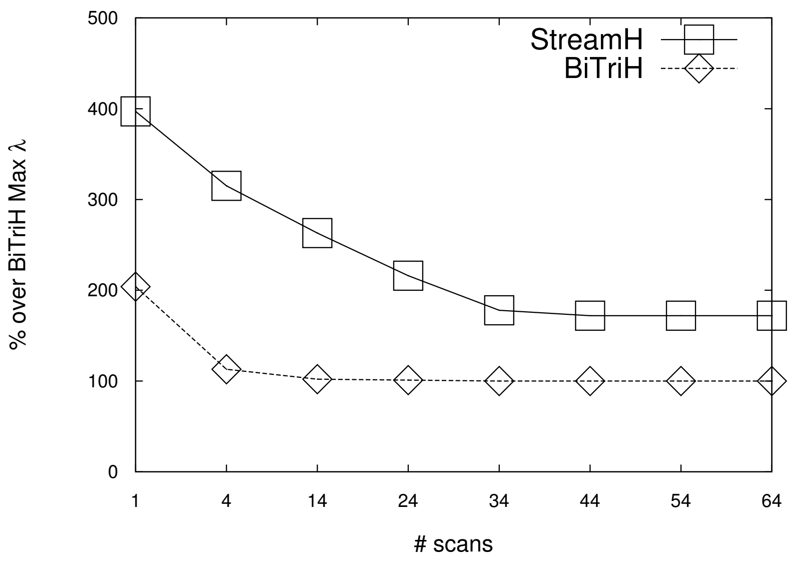


Fig. 5.22: Performance on Flickr Dataset: StreamDN Accuracy

DN-graph Semantics in Various Domain

In the movie co-comments network, two movies are connected by a weighted edge if these two movies share enough commenters such that the Jaccard Coefficient of the two movies' commenters sets is above a certain threshold. Figure 5.23 shows a set of movies and their interactions found in 100 movies co-comments network with a threshold of 0.5. These movies are reported with $\lambda = 9$. All of the 9 movies have exceptional high IMDB scores (> 8 out of 10, which means they are exceptionally popular). Besides the popularity, we found 7 of them are from USA, while the remaining 2 are from France and Japan respectively. The 9 movies all belongs to genre Violence/Fantasy.

Many proteins are functional only when they are assembled into a protein com-

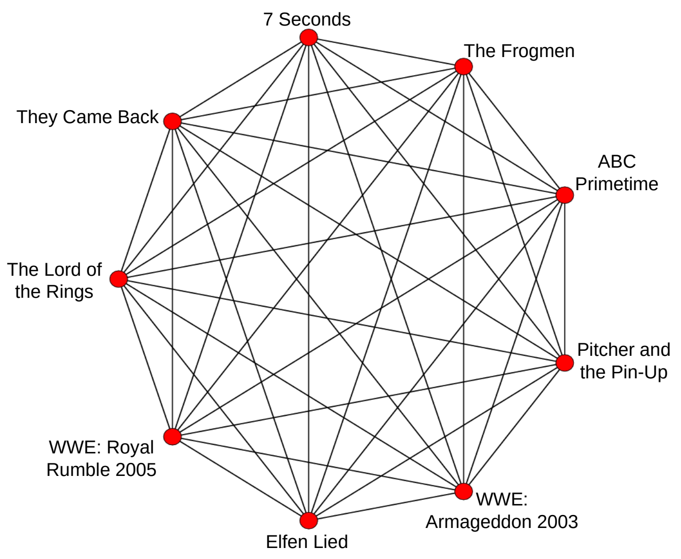


Fig. 5.23: Patterns Discovered in NetFlix

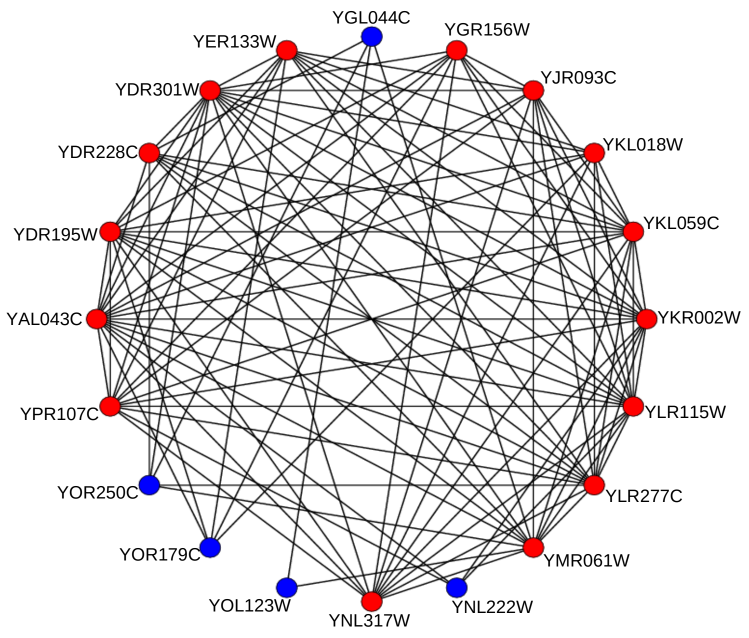


Fig. 5.24: A 20-Protein Complex in Form of DN-Graph

plex. Our DN-Graph mining algorithm can detect important protein complexes out of large amount of protein-protein interaction (PPI) data. For example, mR-

NA_CF_Complex in figure 5.24 is a 20 protein complex confirmed by the benchmark. The red colored nodes are active proteins that functionally interact with other proteins within the complex. These proteins can be successfully detected as an DN-Graph by our algorithm, while others (blue-colored nodes) are missed out in our results. The reason for missing out those proteins is because these proteins have fewer interactions with the rest of the proteins. Even with those missing points, our results are already a significant improvement over known results.

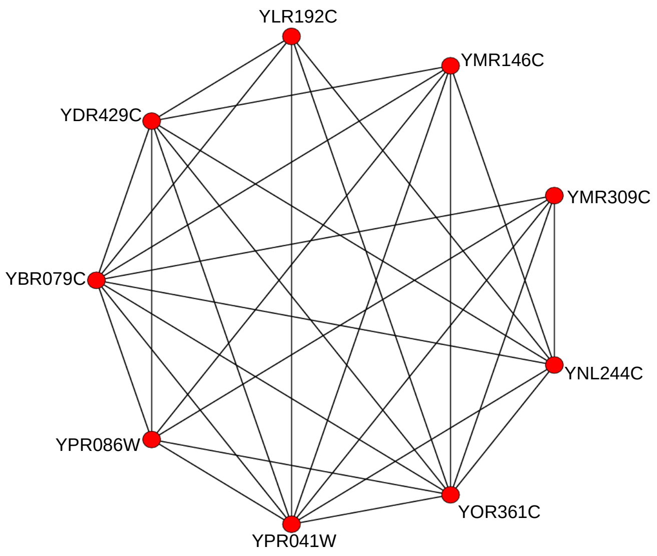


Fig. 5.25: 9-protein exact match

Figure 5.25 and figure 5.26 shows two additional DN-graphs identified by DN-graph mining algorithm. Figure 5.25 is an exact 9-protein complex matched by our algorithm from PPI dataset. Besides identifying known protein complex, DN-graph mining results can also predict unknown proteins' functionality. These proteins are not included in the existing protein complexes benchmark due to sys-

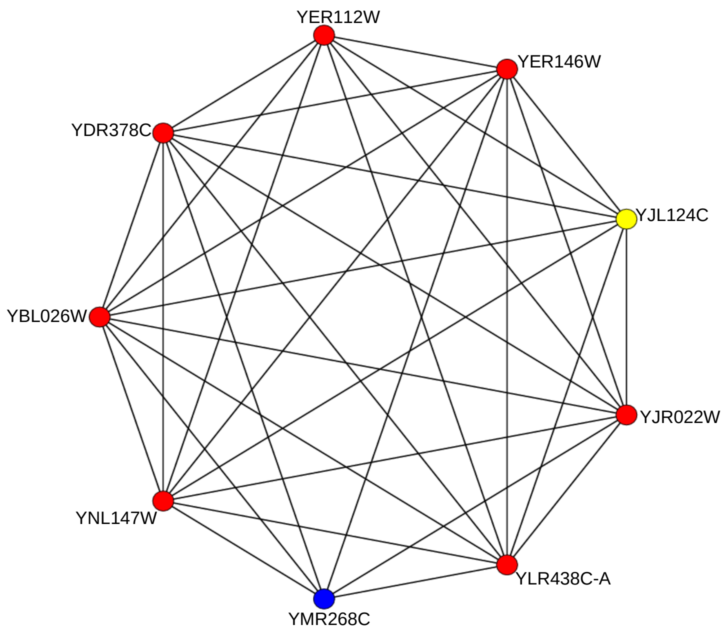


Fig. 5.26: snRNP

tematic experimental constraints. However, by constructing PPI across different experimental sources, this unknown protein may be included into a synergetic DN-graph. In figure 5.26, protein YJL124C was predicted to have the same functionality as the rest proteins inside the graph. We could not find supporting evidence in the existing snRNP protein complex benchmark. However, by checking other domain experts, we confirmed the correctness of our finding. Note that some protein (such as YMR268C) could not be detected due to low connection with other member proteins inside the complex. However, with the observation of high connectivity among the rest proteins, we strongly urge biologists to experimentally search for the “missing“connections between missed proteins and the rest protein members. (For clarity, matches are marked as red points; miss are marked as blue points. Proteins that are not presenting in known benchmark pattern but discovered as members of a DN-graph are marked as yellow.)

5.8 Chapter Summary

In this paper, we present a new graph dense structure DN-graph. The DN-graph complements popular dense structures by imposing both size and degree constraints. We then discuss the graph local triangulation problem and its connection with DN-graph mining problem. Based on that, we propose solutions to effectively locate DN-graphs. The solutions are set of algorithms catering for different problem settings from in memory to streaming. The iterative, triangulation based solution has the advantages that the details can be abstracted within the triangulation algorithm. Since the algorithm improves the result at every iteration, users can stop the algorithm at any time and get the best results within the time limit. Our experimental study shows our solutions are both time and space efficient.

6

DVIG: On-Demand Visualization of Graph Patterns

In previous chapters (Chapters 4 and 5) we address the dense pattern mining problem from the algorithm perspective. The algorithms we proposed help in finding dense patterns. Both a dense pattern's internal structure and its external relationships with other patterns contain important information. It is thus critical to understand these relations. The dense patterns' inter and intra relationships are better to be represented visually for more intuitive understanding.

To help the visualization of patterns, we developed DVIG. DVIG is a lightweight graph pattern visualization tool. With its help, domain experts can visually interpret the inter and intra relationships among graph patterns. DVIG includes the following components (1) a visualization frontend. Users can use it to explore the graph pattern summarization and zoom in to a subset of graph patterns. The visualization frontend also has the functionality to present corresponding subgraphs of the selected patterns. (2) A pattern preprocessor. It is the interface between the D-VIG frontend and the external mining algorithms and (3) a dynamic force-directed layout module. It arranges the pattern subgraphs into an intuitive way. The DVIG system also includes features to present patterns' semantic information. In this demonstration, we showcase the display and exploration of The Digital Bibliography & Library Project (DBLP) network, thereby explaining the architecture and features of the DVIG system.

This chapter consists of four parts. Firstly, in Section 6.1, we address the critical role of a visualization tool plays in graph mining process. Being aware of the importance, we design and develop the DVIG. In this section, we also highlight the DVIG's technical contributions. Secondly, in section 6.2, we present a running example of using DVIG to visualize graph patterns and their semantics. The three main components in the DVIG system and their functionalities are presented along the running example. Thirdly, we describe a demonstrative plan in section 6.4. We end this chapter with a brief summary of the DVIG system in chapter 6.5.

6.1 Visualization Systems are Critical in Graph Mining Process

More and more research attentions have been on visualization of graph mining results. Recently there are many systems developed to address this issue[CFZ06, RJTe06, CFZ06]. In these systems, the discovered patterns are presented in graphical manifests such as charts and diagrams. Aided by these graphical representations, domain experts can understand the semantics of discovered patterns better and these patterns' structures are revealed in a more intuitive manner. Such system further assists experts in deciding the appropriate graph parts which require further analysis. the further analysis is confined into the most necessary parts. The data analysis cost can be optimized via graph mining tools.

We can also use visualization system to explain the intuition behind graph mining algorithms. For example, in previous chapter (Chapter 5), the DN-graph mining algorithm iteratively searches for graph patterns. The patterns are quickly located in the first iteration and are refined in the following iterations. If there is a tool that can visualize the iterative process, users of the mining algorithms are more convinced, as they can "see" the mining process.

In this demonstration, we showcase the **DVIG** On-Demand visualization systems using a running example. In this example, a fresh PhD student of computer science strives to get the most "comprehensive" understanding of how researchers collaborate with each others. However, facing the large volume of citation data, he can hardly read all papers in DBLP to analyze the citation relationship. The DVIG system can help him with above task. By using the DVIG, not only does he identify prominent researchers and research groups in computer science, he also understands the collaboration among groups. The DVIG presents the patterns in an intuitive manner. The DVID's technical contributions include:

1) An intuitive summarization of discovered graph patterns. Being an effective visual tool, it is not sufficient to only visualize patterns individually, since the patterns may be overwhelmingly numerous. A wiser choice is to profile all interesting patterns and present the meta information first before dill down into specific graph patterns. Preferably, The meta information should include an indicative measurement of a pattern's significanceF. The meta information can also guide system users to further investigate those important patterns. With the summarization, domain experts are able to investigate patterns discovered by advanced graph mining algorithms and the complex mechanisms of the algorithms are not visible to them.

2) An layout scheme that organizes the discovered patterns into a force - directed structure. This structure captures the inter and intra relationships among discovered patterns.

6.2 The DVIG Visualization Paradigm

6.3 Visualization Frontend

DVIG visualizes graph patterns from three perspectives: Pattern Summarization Context, Summarization Zoom-In and Pattern Subgraph View. Figure 6.1 is the overview of how the DVIG console displays DBLP authorship patterns. At the bottom left is the pattern summarization. The summarization presents discovered patterns' distribution. Depending on users' choice, the DVIG system is able to zoom into the selected patterns' summarization at the top left area of the console. The corresponding subgraphs are displayed at the right side of the console.

Pattern Summarization Context DVIG summarizes the discovered graph patterns in the form of a **2D plot**. The X-axis represents distinctive graph vertices inside mined graphs, while the Y-axis measures the "connectivity"[1] between this vertex and its right neighbor vertex along the X-axis. The "connectivity" indicates the interesting level of the relationships in which two vertices both participate. The mining results of an algorithm is visualized as a plotting line in the 2D plot.

Depending on the adopted mining algorithms, the lines' shapes vary. If we use DN-graph mining algorithm as the back end algorithm for the DVI, the re-

[1] The definition of "connectivity" varies from graph mining algorithms to algorithms. DVIG however is not affected by such variance. It only requires the "connectivity" definition follows certain partial order.

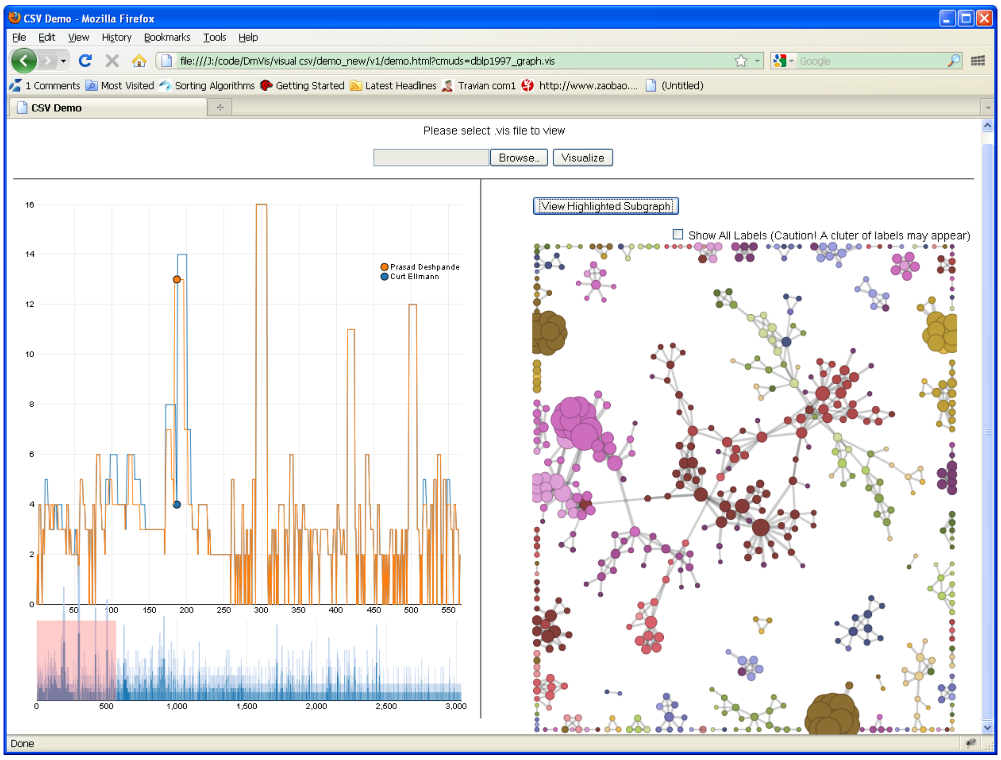

CSV Demo - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///J:/code/DmVis/visual csv/demo_new/v1/demo.html?cmuds=dblp1997_graph.vis
CSV Demo
Please select .vis file to view
Browse...
Visualize
Prasad Deshpande
Curt Ellmann
View Highlighted Subgraph
Show All Labels (Caution! A cluter of labels may appear)
Done

Fig. 6.1: The DVIG Console

sulting plotting lines show “peaks” (i.e., consecutive regions with locally highest “closeness” measure). As discussed previously in Chapter 5, the peaks represents DN-graph patterns, since a vertex is surrounded by its neighbors having the highest “density”.

Moreover, the patterns having smaller distance along the X-axis indicates they are more relevant to each other. This arrangement ensures the 2D plot captures both patterns and their inter connections. By selecting consecutive regions, the domain experts can conveniently select patterns with high relevancy.

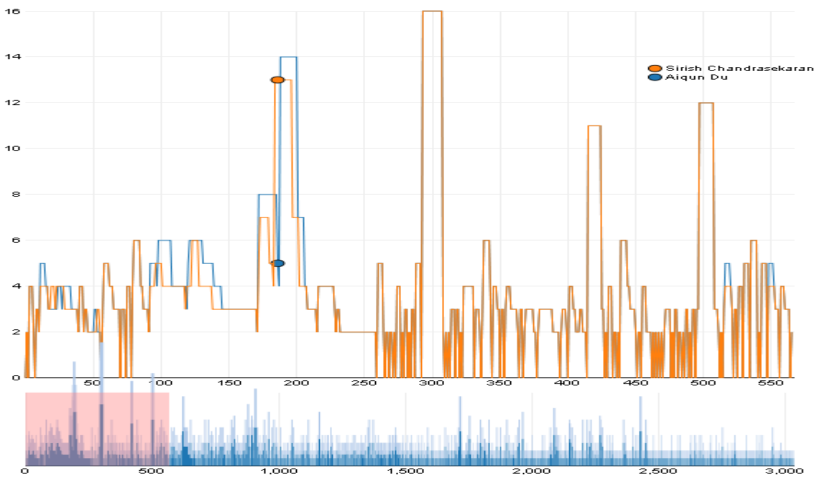
From figure 6.1, we notice there are two plotting lines with similar shapes. The two line represent the first and last iteration’s result (after program converges) of triangulation based DN-Graph mining algorithm 5.2. The two plotting lines have similar shapes. The vertex connectivity measures of two lines differ only slightly. This reassures that the algorithm can produce high quality results in the first iteration.

With Pattern Summarization visualization stated above, system users can further decide which patterns are more interesting to them. In the analysis process, they perform operations such as drilling-down to part of the summarization to have an detailed view of these more interesting patterns.

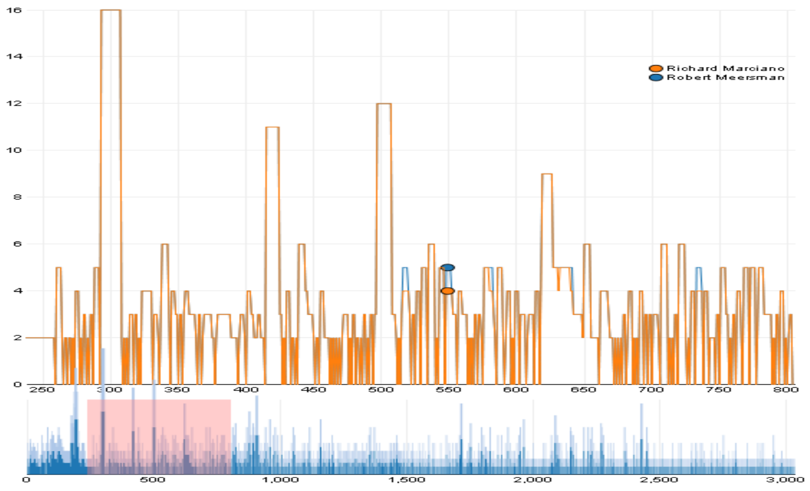
Summarization Zoom-In

After getting an overall summarization, users may further drill down to part of the summarization to investigate interesting pattern. We incorporate a summarization zoom-in area into DVIG. Besides the magnified plot, users can also mouse over this sub area to reveal vertex semantics. We can zoom-in to any consecutive regions of the 2D plot by sliding or resizing the pink selected region. The summarization zoom-in is displayed at the top left of the DVIG console. Figures 6.2(a) and6.2(b) gives two zoom-in views on running example.If we compared the two figures, the bottom summarization views are the same, except the focus areas (in shaded rectangles) are different. At the top of each figure, the magnified views of focus areas are presented to the users.

DVIG can also show vertex label when a user puts his mouse over the zoom-in view. As shown in figures 6.2, the authors names are shown at the top right corner



(a) Example 1



(b) Example 2

Fig. 6.2: The DVIG Pattern Summarization Zoom-In

of the zoom-in view. The colored dot indicates the correspondence between the plotted line and the author name.

After having a zoom-in view of graph patterns, the users usually would like to know the patterns' graph structures. The DVIG also provides another view: pattern subgraph view, to meet this requirement.

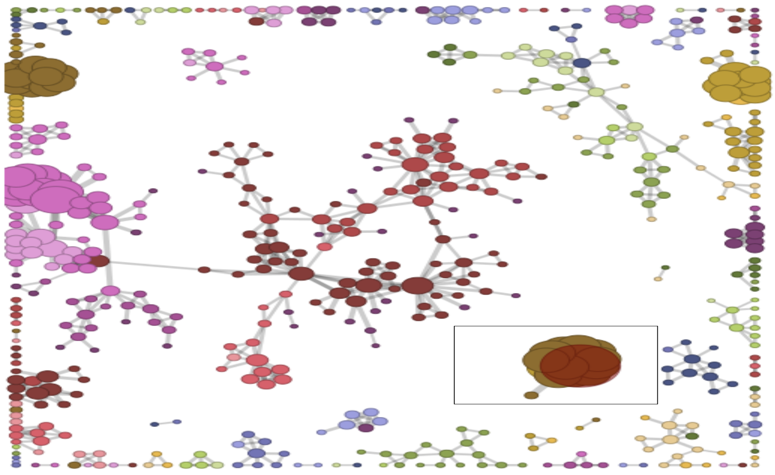
Pattern Subgraph View

The patterns' structure helps users in understanding the pattern semantics better. With appropriate layout scheme, the structure intuitively shows relationships within patterns and among patterns. In the running example below, After zoom-in to the plot, We can choose to view the corresponding co-authorship subgraphs formed by the selected authors. Figure 6.3 shows the subgraph view of Pattern summarization Zoom-In in figure 6.2(a). As the pattern subgraph view can zoom-in and out, we can investigate the subgraphs inside the rectangle in figure 6.3(a) in a more refined scale as in figure 6.3(b).

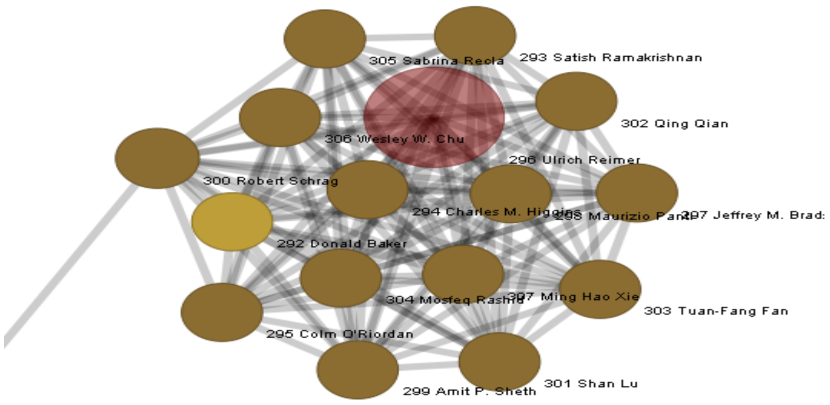
In additional to above three console components, there is one important sub-unit behind scene: The pattern preprocessor.

6.3.1 Pattern Preprocessor

Pattern preprocessor transforms the patterns produced by various graph mining algorithms into a unified internal representation. The representation is an linear order of distinctive vertices, according to their relationships within graph pattern-s. In DVIG, the pattern preprocessor is designed as an layer with the flexibility of importing results from other mining algorithms. This preprocessor may al-so facilitate cross-algorithm comparative studies. For demonstrative purpose, we implement the pattern preprocessor instance that can plug-in into triangle based DN-graph mining algorithm [WZTT11]. The DVIG, however is not restricted



(a) Example 1



(b) Example 2

Fig. 6.3: The DVIG Pattern Subgraph View and Zoom-In

to any specific graph mining algorithms. With other instances of preprocessors, heterogenous mining algorithms can be incorporated into the DVIG.

6.3.2 Dynamic Layout Engine

The dynamic layout engine provides system users with intuition behind graph patterns. The spring-like force are assigned to graph edges and the magnitude of the force is proportional to the number of common neighbors two end vertices share.

With this force-direct layout, system users can have a more intuitive feeling how these patterns are formed. We also color the vertices inside the layout such that vertices from the same pattern have the same color. This brings more intuition in additional to the layout.

6.4 Demonstration Overview

We plan to showcase the features of the DVIG system through a demonstration scenario that visualizes the dynamic citation patterns discovered from the dblp co-authorship graph. In this scenario, a researcher wants to know what are the active research groups.

We demonstrate DVIG constructs Pattern Summarization Context, Summarization zoom-in view and pattern subgraph view for the dblp co-authorship networks via the visualization paradigms described in section 6.2. We illustrate the system by selecting different subset of patterns and visualize corresponding subgraphs in the DVIG.

6.5 Chapter Summary

This chapter presents DVIG, a graph mining pattern visualization paradigm that captures the distribution of graph patterns. Inspired by the need of showing the intuition behind graph patterns, DVIG is designed as an platform to incorporate heterogenous graph mining algorithms and visualize patterns' inter and intra relationships.

DVIG includes two special components (1) A pattern preprocess that integrates mining algorithms's outcomes via a layer of extensible interfaces and (2) a force-direct layout that dynamically shows the formation of graph patterns. Additionally, DVIG supports semantic exploration of graph patterns, by the means of color and vertex labels. In this demonstration, we showcase the DVIG system running on the DBLP co-authorship graph.

7 Conclusion and Future Work

Technological advance has made the collection of large volumes of graph data possible in many domains. How to find important graph patterns becomes a demanding task. Along the progress in graph mining, researchers reconcile that dense patterns have various implications across heterogenous domains, such as social networks, bio-informatics etc. To discover dense patterns out of large graphs, we need to overcome challenges such as: 1. how to decide whether a subgraph is a dense pattern, efficiently; 2. when the graph size is extremely large, how to minimize computational cost; and 3. Last but not lest, how to present the findings in an interpretable way. In this thesis, we provide solutions to graph dense pattern mining and visualization by addressing above challenges. Below is a summarization of the contributions and results of this thesis.

To decide whether a subgraph is a dense pattern efficiently, we provide a density upper bound for each dense pattern. If we arrange graph vertices into a linear order according to this upper bound , we can find all locally maximized fully-connected subgraphs (closed cliques). Further more, the upper bound can substantially reduce search space when searching for exact dense patterns such as closed cliques.

Based on this upper bound, We design a novel algorithm called CSV. It generates an ordering on the vertices of a graph. To quickly compute the upper bound, in CSV, we apply a novel mapping that transforms graph elements (vertices and edges) into high-dimensional points. Existing spatial indices such as the R-tree can be applied to the transformed points, more efficient mining is thus possible. What's more, CSV produces an linear ordering of graph vertices, This order can be used to visualize dense patterns and their distributions. We evaluate CSV on real datasets drawn from stocks correlation networks and DBLP co- authorship networks. The results show that our algorithm is especially useful to locate dense patterns and show their relationships. We also demonstrate the algorithm's effectiveness and efficiency by comparing it with other state-of-the-arts algorithms. In addition to using CSV as a stand-alone tool for visual exploration of dense subcomponents within large graphs we find that it can also be effectively used as a pre-filtering step to significantly speed up exact clique finding algorithms such as

CLAN[WZZ06].

To provide mining solutions for large scaled graph, so that dense pattern mining can carry out within reasonable time and storage constraints, we propos a triangulation-based solution. Inside the iterative, triangulation-based approach, most of the details involved in efficient processing like minimizing I/Os etc., are abstracted within the triangulation algorithm. As the estimation becomes more accurate in every iteration, users can obtain the most updated results at any instance during the course of algorithm running. Further more, when the graph is too large to fit into main memory, statistics can be collected in the first iteration to support effective buffer management should there be a need to store the local density value on a disk, since the triangles come in the same ordering in every iteration.

These set of triangulation based algorithms are for different dense pattern mining settings, ranging from in-memory to disc based graphs, from static to dynamics. We also conduct extensive experiments on several synthetic and factual data sets such as those abstracted from Flickr, the well-known photo sharing network. The experiments show that triangle based solution has more flexibility and effectiveness when handling large scaled graphs.

In additional to algorithms, we need to present the discovered dense patterns in a meaningful way. We hope to uncover knowledge from the complicated internal structure and its relationship with other patterns of a dense pattern. The immediate action towards the discovered patterns is to organize them into a human interpretable way. An analyst organizing the patterns should possess domain knowledge as well as understand the mining results. In fact, we can lighten his load by using an effective mining visualization tool.

The DVIG system is designed to help humans in better interpreting graph mining results. With its assistance, domain experts are able to view the summarization as well as the structures of individual graph patterns. To better reveal the structures, We provide an layout scheme that organizes the structure of discovered patterns into a force-directed way. In additional to that, we incorporate features to display semantics when visualizing domain data in DVIG .

7.1 Future Work

There are several extensions we can continue for mining dense graph patterns. When searching for dense patterns using CSV algorithm, the handling and use of pivots can be extended in at least two directions. First, since the selection of the pivots is done initially without a good understanding of the distribution, refinement of pivots selection could be done after the CSV plot is available. Intuitively, if a pivot is selected from a highly connected region in the graph, its shortest path distances to other vertices in the highly connected region will be short, making it difficult to separate these vertices apart after the mapping. One can also take advantage of spectral plots in this regard. As such, reselecting pivots from less dense regions of the CSV plot could serve to improve the quality of the plot. Second, as mentioned earlier, it may make sense to add in additional pivots when there is a need to hone in on smaller subgraphs.

The handling of directed graph could be useful for some applications like keyword search [HGP03, HP02, BHN$^+$02] where we want to measure the connectivity between keywords. Applying CSV on a directed graph is more complicated in the following ways. Firstly, vertices might not be reachable from the pivots selected. This can be overcome by adding virtual root node to the graph using techniques described in [SU06]. Secondly, after mapping the edges into the high dimensional space, we must record their directions within the grid cell (i.e. the vertex it connects to) and take them into account when computing connectivity. The details of such an approach will be ironed out as part of our future work.

Bibliography

[ABC$^+$04] P. Aloy, B. BãPttcher, H. Ceulemans, C. Leutwein, C. Mellwig, S. Fischer, and A.C. Gavin. In *Structure-Based Assembly of Protein Complexes in Yeast*, volume 303, pages 2026–2029, 2004.

[ABKS99] M. Ankerst, M. M. Breunig, H-P. Kriegel, and J. Sander. OPTICS: Ordering points to identify the clustering structure. In *SIGMOD'99*, pages 49–60, Philadelphia, PA, June 1999.

[ARS02] J. Abello, M.G.C. Resende, and R. Sudarsky. Massive quasi-clique detection. In *In Proc. 5th Latin American Symposium on Theoretical Informatics*, pages 598–612. Springer Verlag, 2002.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB'04*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[ATH03] I. Akihiro, W. Takashi, and M. Hiroshi. In *Complete Mining of Frequent Patterns from Graphs: Mining Graph Data*, volume 50, pages 321–354, Hingham, MA, USA, 2003. Kluwer Academic Publishers.

[AUS07] S. Asur, D. Ucar, and P. Srinivasan. An ensemble framework for clustering proteincprotein interaction networks. In *ISMB'07*, Vienna, Austria, 2007.

[Bas94] D.A. Basin. A term equality problem equivalent to graph isomorphism. In *nformation Processing Letters*, volume 51, pages 61–66, 1994.

[BBCG08] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD'08*, pages 16–24, New York, USA, 2008.

[BBP06] V. Boginski, S. Butenko, and Pardalos. P.M. Mining market data: a network approach. *Computers and Operations Research*, 33(11):3171–3184, 2006.

[BC96] M. Brockington and J. C. Culberson. Camouflaging independent sets in quasi-random graphs. In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *dimacs*, pages 75–88. American Mathematical Society, 1996.

[BHN$^+$02] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *KDD'02*, pages 431–440, Edmonton, Alberta, Canada, 2002.

[Bla94] R.E. Blake. In *Partitioning Graph Matching with Constraints*, volume 27, pages 439–446, 1994.

[BNC03] B. Bustos, G. Navarro, and E. Chávez. In *Pivot selection techniques for proximity searching in metric spaces*, volume 24, pages 2357–2366, New York, USA, 2003. Elsevier Science Inc.

[Bol78] B. Bollobas. *Extremal Graph Theory*. Dover Publications, Incorporated, 1978.

[CFZ06] D. Chakrabarti, C. Faloutsos, and Y.P. Zhan. In *Visualization of Large Networks with Min-cut Plots, A-plots and R-MAT*, 2006.

[CT96] J. Cheriyan and R. Thurimella. Fast algorithms for k-shredders and k -node connectivity augmentation (extended abstract). In *ACM Symposium on Theory of Computing*, pages 37–46, 1996.

[CTTP04] G. Cong, K.-L. Tan, A.K.H. Tung, and F. Pan. Mining frequent closed patterns in microarray data. In *ICDM'04*, pages 363–366, Washington, DC, USA, 2004. IEEE Computer Society.

[CTTX05] G. Cong, K.-L. Tan, A.K.H. Tung, and X. Xu. Mining top-k covering rule groups for gene expression data. In *SIGMOD'05*, pages 670–681, Chicago, IL, USA, 2005. ACM.

[CTX$^+$04] G. Cong, A.K.H. Tung, X. Xu, F. Pan, and J. Yang. Farmer: finding interesting rule groups in microarray datasets. In *SIGMOD'04*, pages 143–154, Paris, France, 2004. ACM.

[DBLEH07] Skip Farmer David B Little and Oussama El-Hilali. *Digital Data Integrity: The Evolution from Passive Protection to Active Management*. Wiley, 2007.

[Der03] S. Deroski. Multi-relational data mining: an introduction. *SIGKDD Explorations Newsletter*, 5(1):1–16, 2003.

[DT99] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(7-36), 1999.

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. In *KDD'96*, pages 226–231, Portland, Oregon, Aug. 1996.

[FL95] C. Faloutsos and K.-I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *SIGMOD'95*, pages 163–174, San Jose, CA, May 1995.

[FTCF01] R.S. Filho, A. Traina, T.Jr. Caetano, and C. Faloutsos. Similarity search without tears: The omni family of all-purpose access methods. In *ICDE'01*, Heidelberg, Germany, 2001.

[GE02] A.P. Gasch and M.B. Eisen. Exploring the conditional coregulation of yeast gene expression through fuzzy k-mean clustering. *Genome Biology*, 3(RESEARCH 0059), 2002.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: a Guide to The Theory of NP-Completeness*. Freeman and Company, New York, 1979.

[GRT05] D. Gibson, K. Ravi, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB'05*, pages 721–732, Trondheim, Norway, 2005.

[GS05] A. Gulli and A. Signorini. In *The Indexable Web is More than 11.5 Billion Pages*, Chiba, Japan, 2005.

[HCD94] L. Holder, D. Cook, and S. Djoko. Substructure discovery in the SUBDUE system. In *Proceedings of the Workshop on Knowledge Discovery in Databases*, pages 169–180, 1994.

[HGP03] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB'03*, pages 850–861, 2003.

[HK00] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[HMWD04] Z. Hu, J. Mellor, J. Wu, and C. DeLisi. VisANT: an online visualization and analysis tool for biological interaction data. In *BMC Bioinformatics*, volume 5, pages 17–24, 2004.

[HP02] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *VLDB'02*, pages 670–681, 2002.

[HW74] J.E. Hopcroft and J.K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *STO'74*, pages 172–184, 1974.

[HYH$^+$05] H. Hu, X. Yan, Y. Huang, J. Han, and X.J. Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. In *Bioinformatics*, volume 1, pages 1–1, 2005.

[Inc10] Yahoo! Inc. Flickr - photo sharing. `http://www.flickr.com/`, 2010. Online; accessed 20-Dec-2010.

[KH04] E.B. Krissinel and K Henrick. In *Common subgraph isomorphism detection by backtracking search*, volume 34, pages 591 – 607, 2004.

[KI02] H. Kashima and A. Inokuchi. Kernels for graph classification. *Proceeding of International Workshop on Active Mining*, 2002.

[KV96] G. Karypis and K. Vipin. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, page 35, Washington, DC, USA, 1996. IEEE Computer Society.

[KW06] G. Kossinets and D. J. Watts. Empirical analysis of an evolving social network. *Science Magazine*, 311(5757), 2006.

[Lat07] M Latapy. Practical algorithms for triangle computations in very large (sparse (power-law)) graphs. volume 407 (1-3), pages 458 – 473, 2007.

[Luk82] E.M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer System Science*, pages 42–65, 1982.

[MARW90] E.M. Mitchell, P.J. Artymiuk, D.W. Rice, and P. Willett. Use of techniques derived from graph theory to compare secondary structure motifs in proteins. *Journal of Molecular Biology*, 212:151–166, 1990.

[MB00] B.T. Messmer and H. Bunke. Efficient subgraph isomorphism detection: A decomposition approach. *TKDE'00*, 12(2):307–323, 2000.

[MK01] K. Michihiro and G. Karypis. Frequent subgraph discovery. In *ICDM'01*, pages 313–320, 2001.

[net] Netflix prize data set. `http://www.netflixprize.com/`. [Online; accessed 20-March-2010].

[NJW01] A.Y. Ng, M.I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, volume 14, 2001.

[NK99] S. Nijssen and J. Kok. Fast association rules for multiple relations. volume 3, 1999.

[PCT$^+$03] F. Pan, G. Cong, A.K.H. Tung, J. Yang, and M.J. Zaki. Carpenter: finding closed patterns in long biological datasets. In *KDD'03*, pages 637–642, Washington, DC, USA, 2003. ACM.

[PTCX04] F. Pan, A.K.H. Tung, G. Cong, and X. Xu. Cobbler: Combining column and row enumeration for closed pattern discovery. In *SSDBM '04: Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, page 21. IEEE Computer Society, 2004.

[RJTe06] J.F. Rodrigues Jr. and H.H. Tong etc. GMine: a system for scalable, interactive graph visualization and mining. In *VLDB'06*, pages 1195–1198, Seoul, Korea, 2006. VLDB Endowment.

[RRRT99] K. Ravi, Prabhakar R., Sridhar R., and A Tomkins. Trawling the web for emerging cyber-communities. In *Computer Networks*, pages 1481–1493, 1999.

[Sco00] J. Scott. *Social network analysis: A handbook.* Sage, 2000.

[Sei83] S.B. Seidman. Network structure and minimum degree. *Social Networks*, 5:269–287, 1983.

[SK98] A. Srivastav and W. Katja. Finding dense subgraphs with semidefinite programming. In *APPROX '98*, pages 181–191, London, UK, 1998. Springer-Verlag.

[SMT91] J.W. Shavlik, R.J. Mooney, and G.G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6:111–144, 1991.

[SU06] T. Silke and L Ulf. GRIPP - indexing and querying graphs based on pre and postorder numbering. Technical report, 2006.

[SW05] T Schank and D. Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *WEA*, pages 606–609, 2005.

[Tur41] P. Turan. On an extremal problem in graph theory. *Mat. Fiz. Lapok*, 48:436–452, 1941.

[Ull76] J.R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976.

[Vap95] V.N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995.

[Wik06] Wikipedia. Protein protein interaction — Wikipedia, the free encyclopedia, 2006. [Online; accessed 1-May-2010].

[WM03] T. Washio and H. Motoda. In *State of the Art of Graph-based Data Mining*, volume 5, July 2003.

[WSTT08] N. Wang, P. Srinivasan, K.-L. Tan, and A.K.H. Tung. CSV: visualizing and mining cohesive subgraphs. In *SIGMOD'08*, pages 445–458, 2008.

[WZTT11] N. Wang, JB. Zhang, K.-L. Tan, and A.K.H. Tung. On triangulation-based dense neighborhood graph discovery. In *VLDB'11*, volume 4, 2011.

[WZZ06] J. Wang, Z. Zeng, and L. Zhou. CLAN: An algorithm for mining closed cliques from large dense graph databases. In *ICED'06*, page 73, 2006.

[XSe02] I. Xenarios and Lukasz. Salwinski etc. DIP, the database of interacting proteins: A research tool for studying cellular networks of protein interactions. *Nucleic Acids Research*, 30(1):303–305, 2002.

[YH02] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the International Conference on Data Mining*, pages 721–724, 2002.

[YMI94] K. Yoshida, H. Motoda, and N. Indurkhya. Graph based induction as a unified learning framework. In *Applied Intelligence*, volume 4, 1994.

[YZH05] X. Yan, X.J. Zhou, and J. Han. Mining closed relational graphs with connectivity constraints. In *KDD'05*, pages 324–333, Chicago, IL, USA, 2005.

[ZWZK06] Z. Zeng, J. Wang, L. Zhou, and G. Karypis. Coherent closed quasi-clique discovery from large dense graph databases. In *KDD'06*, pages 797–802, Philadelphia, USA, 2006.